31 OPERATORS MANUAL

TEKTRONIX®

CALCULATOR PRODUCTS

All Tektronix instruments are warranted against defective materials and workmanship for one year.

Additionally, all Tektronix Computer Display Terminals and related computer peripheral equipment are fully warranted against ANY trouble for the first 90 days. Any equipment trouble occurring to your Tektronix computer terminal or related products during the 90 day period will be repaired by Tektronix personnel at no charge.

Questions regarding warranty should be discussed with your Applications Engineer.

Specifications and price change privileges reserved.

31 OPERATORS MANUAL

070-1575-00

CONTENTS

1 KEYBOARD

2 PROGRAMMING

CONTENTS

3 GENERAL INFORMATION

4 APPENDICES (tabs)

KEYBOARD

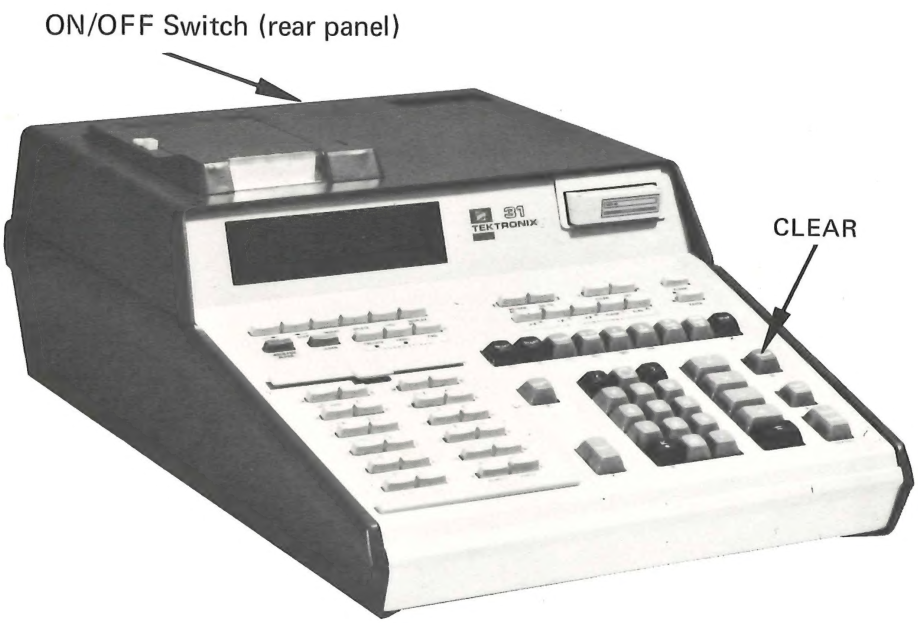
GETTING STARTED

"Mathematics, if it has a function at all in the sciences, has an indispensable one; and now that man's relation to man has come to be treated in mathematical terms, it is impossible to ignore or escape the new language any longer."

— — —ABRAHAM KAPLAN
Sociology Learns the
Language of Mathematics

Everyone uses mathematics — it is a language. You converse with the Tektronix calculator in its language: Mathematics. Everybody knows it. Anyone can do it.

The most commonly used arithmetic operator keys are those which add +, subtract —, multiply X, and divide ÷. These keys perform their indicated operations on successive numbers that are entered into the display. The results are displayed whenever the equal key, =, is pressed.



Turn on your calculator and press the red key labeled "CLEAR" to eliminate the flashing display.

Do a simple addition: 3 + 2 = 5

PRESS $\boxed{3}$ $\boxed{+}$ $\boxed{2}$ $\boxed{=}$

The display reads 0000000005.

We disregard the leading zeros; they don't mean anything.

Now subtract: 6 – 2 = 4

PRESS $\boxed{\text{CLEAR}}$ (get ready)

PRESS $\boxed{6}$ $\boxed{-}$ $\boxed{2}$ $\boxed{=}$

Read the answer, 4, in the display.

Try multiplying: 10 X 5 = 50

PRESS $\boxed{\text{CLEAR}}$

PRESS $\boxed{1}$ $\boxed{0}$ $\boxed{\times}$ $\boxed{5}$ $\boxed{=}$

Read the answer, 50, in the display.

Now divide 10 ÷ 5 = 2

PRESS $\boxed{\text{CLEAR}}$

PRESS $\boxed{1}$ $\boxed{0}$ $\boxed{\div}$ $\boxed{5}$ $\boxed{=}$

Read the answer, 2, in the display.

GETTING STARTED

Now try combining these examples by adding each example.

PRESS [CLEAR]

PRESS [3] [+] [2] [+] [6] [−] [2] [+] [1] [0] [×] [5] [+] [1] [0] [÷] [5] [=]

Read the answer, 61, in the display.

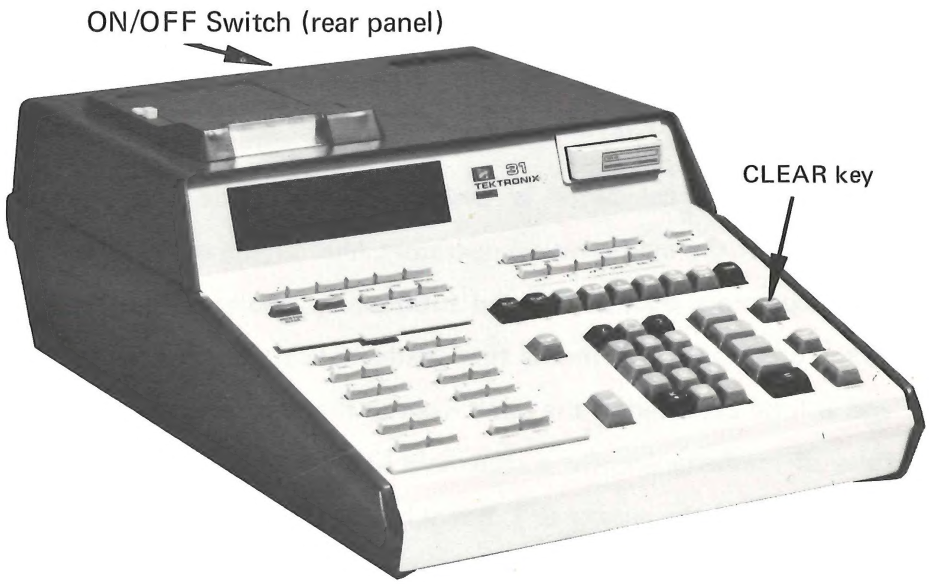
By now you see that anyone with an understanding of the meaning of the symbols +, −, X, and ÷ can use the calculator to add, subtract, multiply, and divide. Don't be confused by the other keys, we'll get to them later. What's important now is that you realize that the machine is designed to accept instructions from you in a conventional manner. The calculator will communicate with you to the limit of your understanding of its language. Mathematics is that language.

A WORD ABOUT HIERARCHY

The Tektronix calculator is designed to recognize the precedence or hierarchy of the mathematical operators. Multiplying and dividing are higher order operators than adding and subtracting. The machine recognizes this fact. To the user this means that mathematical statements can be entered and will be executed in the same manner they are written, according to the rules of mathematics. (In a statement like 1 + 2 X 3, the multiplication is done *first* and the answer is 7, not 9.) A further discussion of hierarchy can be found in the **HIERARCHY** section.

INITIALIZATION

Before the calculator can be exercised with any particular problem it must be prepared to accept your commands. Thus you should "initialize" the machine at the beginning of any problem. The initialization procedure will become automatic to you and involves *turning on the machine, pressing the CLEAR key,* and *possibly changing the trigonometric operating mode from radians to degrees.*



The ON-OFF switch applies AC power to the calculator and is located on the back of the machine, near the top. When the power is first turned on, the calculator will always react in the following manner:

* All programmable memory is cleared. The memory was erased when the power was turned off.

* The machine indicates a memory erasure by flashing the display.

* The calculator is set in the Idle mode. It is waiting for instructions.

* The trigonometric operating mode is set to radians, as indicated by illumination of the RAD light under the display. (This will be discussed in detail in the **TRIGONOMETRIC AND HYPERBOLIC KEYS** section.)

INITIALIZATION

CLEAR, abbreviated CLR, is the red key located on the right of the keyboard. Pressing the CLEAR key prepares the calculator for a new set of calculations. As you become more familiar with the machine you will find that you use this key quite often. When cleared, the calculator reacts in the following manner:

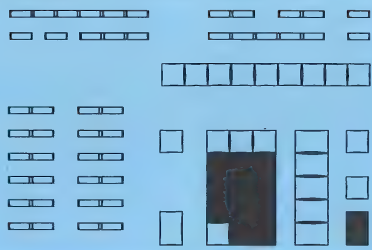
* The display is cleared to all zeros.

* The calculator is reset to establish the starting point for a new calculation.

* Radian, RAD, operation is reestablished if the machine had previously been operating in the degree, DEG, mode.

* Pressing CLR resets the calculator's error message (a flashing display). A flashing display may be caused by:

    1. A power interruption
    2. An illegal operation
    3. An out of range display



KEYBOARD

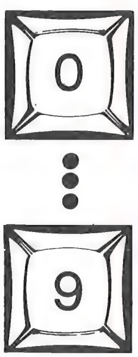
DATA ENTRY KEYS

When using the calculator, you will be entering various numbers that are a part of the particular problem you are solving. We refer to these numerical entries or inputs as 'data'. The data you use will take several forms:

DATA FORMATS

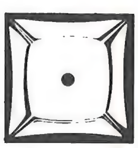
DECIMAL or ORDINARY	100	–512	.0001	243.123
SCIENTIFIC	1×10^{2}	-5.12×10^{2}	1×10^{-4}	2.43123×10^{2}
MIXED	10×10^{1}	-51.2×10^{1}	$.01 \times 10^{-2}$	$.00243123 \times 10^{5}$

Occasionally you will make errors in the keystrokes necessary to enter data and will need to correct the mistakes before proceeding. The data entry keys allow all of the above flexibilities.



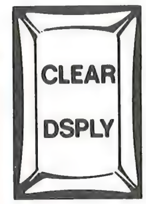
The ten numeric keys, 0 through 9 are arranged in the standard adding machine format. Pressing any of these keys causes the corresponding digit to be shown on the display. Sequentially pressing the various keys results in a serial representation of the keystrokes. The display accepts up to ten digits from the keyboard; further entries have no effect.

While only ten digits can be entered into the display, to maintain accuracy, the calculator performs its computations with twelve digits. Thus, the result of any computation is rounded to ten digits before it is displayed. The two digits which are not displayed are called "guard digits". A further discussion of guard digits is contained in the section **OPERATING CHARACTERISTICS.**

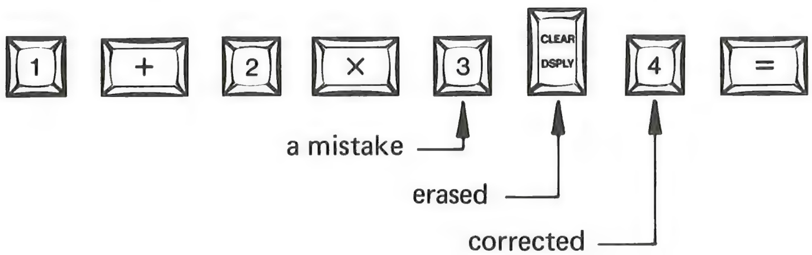


The decimal point key enters a decimal point into the display. Simply press this key at the desired point in any numeric entry sequence. When entering integers, it is not necessary to enter a decimal point; the machine will assume one after the last digit.

DATA ENTRY KEYS



The CLEAR DISPLAY key is different from the red CLEAR key. As implied, the CD key only clears the display, resetting it to all zeros. The CD key is used to erase a numerical entry from the display and is mainly used to correct mistakes in data entry. As distinct from the CLR key, the CD will not change or alter previous data entries or operations – it does nothing more than erase the display. For example, the sequence



will result in the answer, 9, since the CD only eliminates the 3 from the expression.

EXAMPLE: DATA ENTRY

Enter the number 5

PRESS 

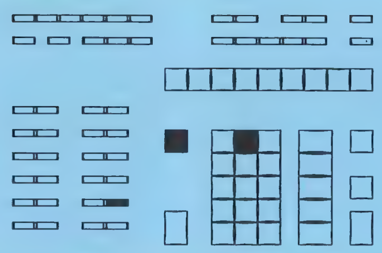
PRESS 

Enter the number 50.123

PRESS 

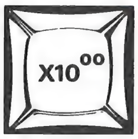
PRESS 

Note the location of the decimal point. Note that the use of CD does nothing more than clear the current contents of the display.



KEYBOARD

DATA ENTRY KEYS



When entering data with exponents in either mixed or scientific notation, the X10⁰⁰ key sets the display to accept entry of any one or two digit exponent.

* If more than two digits are entered for any given exponent, they replace the first and second. Use this feature to correct improperly entered exponents without changing the mantissa. Try pressing

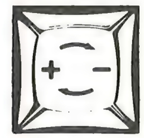
  X10⁰⁰ 1 2 3 4

* Use the +/− key to change the sign of the exponents before or after the exponent has been entered.

* Displayed data in ordinary or mixed notation within the range of the calculator may be changed to scientific notation by use of the X10⁰⁰ = keying sequence.

EXAMPLE: Change 180 to 1.8×10^2

PRESS	CLEAR DISPLY 1 8 0	Display 0000000180
PRESS	X10⁰⁰ =	Display 1.800000000 02



The calculator assumes that any entered number is positive unless otherwise specified. To change the sign of an entry, use the +/– key.

* To change the sign of a mantissa, press +/– before or after entry.

* To change the sign of an exponent, press +/– after $\times 10^{00}$ (see below).

* When +/– is used to change the sign of any exponent or mantissa (except π), the two guard digits are lost. (See **OPERATING CHARACTERISTICS.**)


π

π enters the unrounded twelve digit value of π into the display. Whenever this key is used, it changes any existing display to 3.141592653. Since the display consists of only ten digits, only the first ten digits of π are displayed. The other two digits are stored as guard digits. The rounded value of π is obtained by pressing


π

DATA ENTRY KEYS

THE GOOGOL LIMITATION

A googol is defined as 1×10^{100} or 1 followed by 100 zeros. The range of the Tektronix calculator is limited to slightly less than one googol. Is this really a limitation? How big is a googol?

To answer these questions consider packing small objects into large volumes. Use the following constants:

Earth's radius	=	3963.34 miles	
Silicon crystal	=	10^{22} atoms/cm^3 =	4.2×10^{37} atoms/mi^3
volume of sphere	=	$(4/3)\,\pi r^3$	
one mile	=	5280 ft	
one inch	=	2.54 cm	
velocity of light	=	186,000 miles/sec	

Result: A sphere of crystalline silicon the size of earth would contain about 1×10^{50} atoms, which is 50 orders of magnitude less than a googol.

Result: The volume ratio of a silicon atom to the earth is about the same as the volume ratio of the earth to a spherical googol of silicon atoms. Or, more precisely

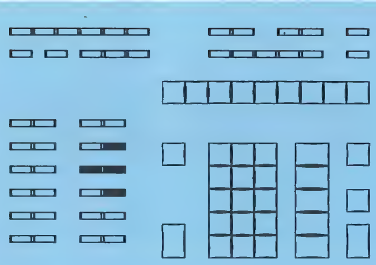
$$\frac{\text{one Silicon atom}}{\text{one earth}} \approx \frac{\text{one earth}}{10^{100}\ \text{Silicon atoms}}$$

The diameter of the silicon sphere containing a googol of atoms exceeds ninety million light-years!

Result: A googol is large indeed. In our world there might not be a googol of anything.

Finally then, it is intuitively difficult to conceive any practical problem that will yield near googol results. As a consequence, a flashing display of all nines on the calculator is usually the result of an illegal math operation. Granted, there are situations which will legally drive the machine into an over-range condition, but careful inspection of a problem will usually resolve these difficulties.

For further information on error messages and over-range conditions, please refer to the section on **OPERATING CHARACTERISTICS.**



MATH KEYS

The single operand function keys are those labeled with functions that require a single data entry.

A single operand key performs its indicated operation on the number that is in the display when the key is pressed, replacing the number in the display with the result of the operation indicated on the key.

$\frac{1}{x}$

1/x will take the reciprocal of the displayed number and display the result.

EXAMPLES:

1. Find the reciprocal of 2.

   PRESS [CLEAR DSPLY] [2] [$\frac{1}{x}$]   Read the answer, 0.5, in the display.

2. Find x = 2/(4 X 5)

   PRESS [CLEAR DSPLY] [4] [×] [5] [=] [$\frac{1}{x}$] [×] [2] [=]

   Read the answer, 0.1, in the display.

3. Find x = 1/0

   PRESS [CLEAR DSPLY] [$\frac{1}{x}$]   FLASHING 9.999999999 99

   The flashing display is the result of an illegal operation since division by zero is undefined. You may wish to use the CD 1/x sequence later when programming to indicate some condition. A CLR eliminates the flashing display.

x^2 squares the number in the display and replaces the number with the result.

x^2

$\sqrt{x}$ computes the square root of the number in the display and replaces the number with the result.

$\sqrt{x}$

EXAMPLE: $5^2 = 25, \sqrt{25} = 5$

PRESS CLEAR DSPLY | 5 | x^2 — Display reads: 0000000025.

PRESS $\sqrt{x}$ — Display reads: 0000000005.

If you attempt to find the square root of a negative number, the calculator will ignore the negative sign and find the square root of the positive number and present the result in a flashing display. (Try calculating the square root of −4.)

int x truncates any display to its integer value. The sign is retained.

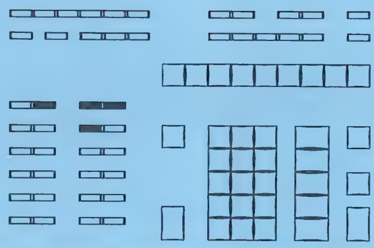
int x

EXAMPLES:

Convert the ordinary, scientific and mixed forms of 123.4 to integer displays.

	Keys	Integer Display
ORDINARY	CLEAR DSPLY, 1, 2, 3, ·, 4, int x	0000000123
SCIENTIFIC	CLEAR DSPLY, 1, ·, 2, 3, 4, x10⁰⁰, 2, int x	1.230000000 02
MIXED	CLEAR DSPLY, 1, 2, ·, 3, 4, x10⁰⁰, 1, int x	1.230000000 02

Note the two forms of integer displays



KEYBOARD

MATH KEYS

logX

logX converts the number in the display to its log (base 10). Note that log (−N) and log (0) result in flashing displays of log (+N) and $-9.999999999 \times 10^{99}$ respectively.

EXAMPLES:

Find $\log_{10} (1.234 \times 10^2) = 2.09131516$

PRESS [CLEAR DSPLY] [1] [.] [2] [3] [4] [×10⁰⁰] [2] [logX]

Find $\log_{10} (1.234 \times 10^{-2}) = -1.90868484$

PRESS [CLEAR DSPLY] [1] [.] [2] [3] [4] [×10⁰⁰] [+/−] [2] [logX]

e^x

e^x raises e, the base of Naperian or natural logarithms, to the power indicated in the display. (e = 2.718281828)

lnX

lnX computes the Naperian or natural log of the display. Again, as with log (base 10), the natural log of a negative number and zero will result in flashing displays.

EXAMPLES:

Compute $e^1 = 2.718281828$, and ln(e) = 1	DISPLAY
PRESS [CLEAR DSPLY] [1] [e^x]	2.718281828
PRESS [lnX]	0000000001.
To compute ln(2)	
PRESS [CLEAR DSPLY] [2] [lnX]	.6931471806

x!

x! calculates the factorial of the number in the display.

EXAMPLE:

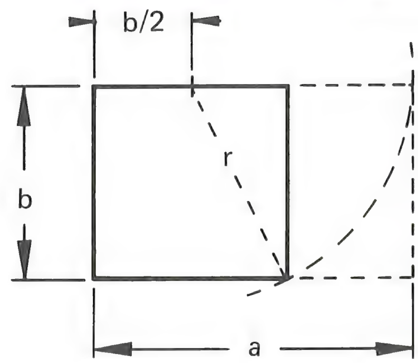
Calculate $4! = 4 \times 3 \times 2 \times 1 = 24$

PRESS [CLEAR DSPLY] [4] [x!]	0000000024.

TEK 31

THE GOLDEN RATIO

In the following figure the golden ratio is defined as a/b.



The solution $\frac{a}{b} = \frac{\sqrt{5}+1}{2}$ has some interesting properties:

PRESS [CLEAR DSPLY] [5] [$\sqrt{x}$] [+] [1] [=] [÷] [2] [=]

DISPLAY 1.618033989

PRESS [$\frac{1}{x}$] .6180339888

Note the equivalence of the decimal portions

PRESS [$\frac{1}{x}$] [x^2] 2.618033989

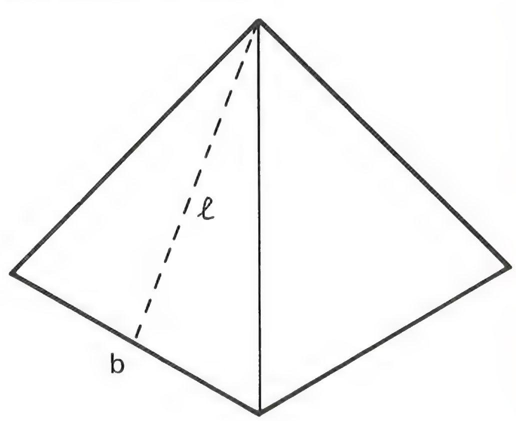
The golden ratio was known and used in the construction of the Great Pyramid about 3000 B.C. Try finding the ratio of the length of one face to one-half the base [ℓ/(b/2)]. Is it an accident that this approximates the golden ratio? What is the relationship between the area of one face (A = 1/2·b·ℓ) and the height squared ? What is the relation between the ratio of the base to the height (b/h) and the constant π?

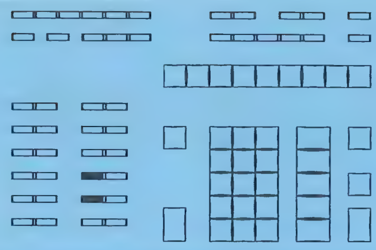
GREAT PYRAMID

height = 481.40 feet

length = 612.00 feet

base = 755.79 feet





MATH KEYS

[ ] $\sqrt{\Sigma x^2}$

$\sqrt{\Sigma x^2}$ as implied will calculate the square root of a sum of squares. In addition to its obvious uses, this key has other applications, for instance, giving the absolute value of a number.

EXAMPLES:

Solve the triangle $X = \sqrt{3^2 + 4^2}$

PRESS [CLEAR DSPLY] [3] [$\sqrt{\Sigma x^2}$] [4] [=]

Did you get the answer, 5?

This key is also very useful for evaluating expressions of the following form

$$X = \sqrt{a^2 + b^2 + \cdots + n^2}$$

For example, to find $\sqrt{3^2 + 4^2 + 5^2}$,

PRESS [CLEAR DSPLY] [3] [$\sqrt{\Sigma x^2}$] [4] [$\sqrt{\Sigma x^2}$] [5] [=]

The result should be equal to $\sqrt{50}$. Is it? (This key is very useful when working with vectors.)

When using this key, if data is not entered, the zero in the display is used.

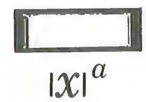
For example, the key sequences

[CLEAR DSPLY] [$\sqrt{\Sigma x^2}$] [5] [+/-] [=] $[\sqrt{0^2 + (-5)^2}\,]$

and

[CLEAR DSPLY] [5] [+/-] [$\sqrt{\Sigma x^2}$] [=] $[\sqrt{(-5)^2 + 0^2}\,]$

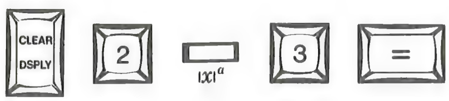
will both produce the same result, 5. (Thus this key is also useful for obtaining the absolute value of any number.)



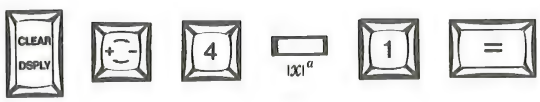
$|x|^a$ raises the *magnitude* of the number in the display to a power subsequently entered. It is also useful in finding absolute values.

EXAMPLES:

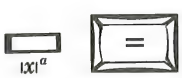
Find $2^3 = 8$

PRESS 

Find $|-4| = 4$

PRESS 

Later, when you are programming, you may find need to test a number to see if it is either zero or non-zero. On the calculator $0^0 = 0$ and hence, with a number in the display, the sequence

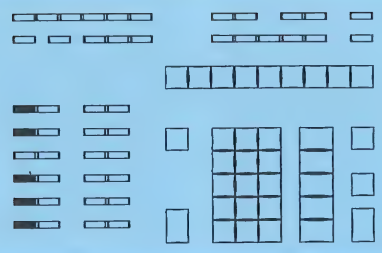


$0^0 = 0$

$N^0 = 1$

will result in a zero if the displayed number was zero or a one if the displayed number was non-zero. (This has natural applications for Boolean Algebra.)

NOTE: Since $|x|^a$ raises the *magnitude* of a number to a power, using this key to raise a negative number to an odd power results in a positive rather than negative number.



KEYBOARD

TRIGONOMETRIC AND HYPERBOLIC KEYS

The calculator is always in one of two possible trigonometric modes: *radians* or *degrees.* The current operating mode is signified by reference to the illuminated status indicators, 'RAD' and 'DEG' under the display. One or the other of these indicators is always on, indicating current status.

DEG RAD

* With the calculator in the 'RAD' mode, pressing the D/R key converts a displayed number in radians to its equivalent in degrees. The machine remains in the Degree mode for all subsequent trigonometric calculations or until the Radian mode is re-established by a second D/R keystroke or a CLR.

* The D/R key allows you to change the form of the displayed number from scientific notation to ordinary notation. With a displayed number in scientific notation, say 1.234×10^3, press D/R twice to convert it to 1234.

Radians and degrees are two forms of angular measurement; there are π radians in 180 degrees.

EXAMPLE:

		DISPLAY	STATUS
PRESS	CLEAR	0000000000	'RAD'
PRESS	π	3.141592653	'RAD'
PRESS	DEG RAD	0000000180.	'DEG'

To convert from scientific to ordinary notation,

PRESS	CLEAR DEG RAD	0000000000	'DEG'
PRESS	1 · 8 $\times 10^{00}$ 2	0000000001.8 02	'DEG'

TRIGONOMETRIC AND HYPERBOLIC KEYS

PRESS	DEG RAD	3.141592654	'RAD'
PRESS	DEG RAD	0000000180.	'DEG'

Unless the operating mode of the calculator is changed to degrees prior to the use of any of the trigonometric keys, all trigonometric calculations are done in radians. If the Degree mode is desired, you may change the trigonometric operating mode with the D/R key. Current trigonometric status is denoted by an illuminated status indicator light located below the display. (Remember, pressing CLR will always restore the calculator Radian mode.)

sin x

cos x

tan x

sin x, cos x, and **tan x** denote the three trigonometric keys for the sine, cosine, and tangent functions. Each of these keys performs the indicated trigonometric operation on the displayed number and replaces it with the result.

arc

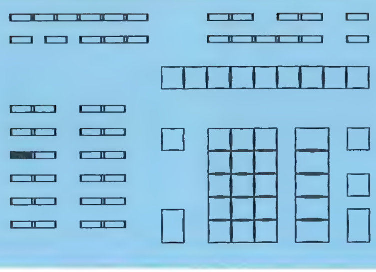
To find the angle that is the inverse trigonometric function of a displayed number, press **arc** followed by the desired trigonometric function.

EXAMPLES:

Find the angle, in radians and degrees, whose tangent is unity.

$$\theta = \tan^{-1}(1)$$

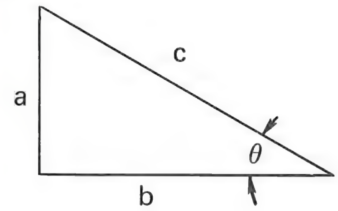
PRESS	CLEAR 1 arc tan x	DISPLAY	.7853981634 radians
PRESS	DEG RAD	DISPLAY	0000000045 degrees



KEYBOARD

TRIGONOMETRIC AND HYPERBOLIC KEYS

Try solving the following triangle for all trigonometric functions of the indicated angle. Exercise the arc key to return to the angle after an operation to correlate the results.



$\sin\theta = a/c$

$\cos\theta = b/c$

$\tan\theta = a/b$

$\tan\theta = a/b,\ \theta = \tan^{-1}(a/b)$

$$\theta = \tan^{-1}\frac{1}{\sqrt{3}}$$

PRESS [CLEAR] [3] [$\sqrt{x}$] [$\frac{1}{x}$] [arc] [tan x]

Thus θ is an angle of .5235987756 radians

PRESS [DEG RAD]

or θ is an angle of 30 degrees

PRESS [sin x]

The sin of 30 degrees is .5

PRESS [arc] [sin x]

An angle whose sin is .5 is 30 degrees.

PRESS [cos x]

The cos of 30 degrees is .8660254038

PRESS [arc] [cos x]

An angle whose cos is .8660254038 is 30 degrees

PRESS [tan x]

The tan of 30 degrees is .5773502691

When calculating inverse trigonometric functions, arc tan or arc sin, the result will always be an angle between −90 degrees ($-\pi/2$) and +90 degrees ($\pi/2$) and when calculating arc cos the results will be an angle between 0 degrees and +180 degrees (π).

TRIGONOMETRIC AND HYPERBOLIC KEYS

hyper

hyper when followed by any of the trigonometric function keys, will convert the display to its hyperbolic function. To obtain the inverse hyperbolic functions the **arc** keystroke may either precede or follow the **hyper** keystroke. Note that the arguments of the hyperbolic functions do not have units and are therefore insensitive to the trigonometric mode (Degree or Radian).

EXAMPLES:

The hyperbolic sin of x is written sinh(x) and is defined by

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

Find the hyperbolic sin for x = .5 by two methods.

METHOD 1: Using the EXPONENTIAL KEYS

PRESS CLEAR . 5 e^x − +/− . 5 e^x = ÷ 2 =

The result is .5210953055

METHOD 2: Using the HYPERBOLIC KEYS

PRESS CLEAR . 5 *hyper* *sin x*

Did you get the same answer?

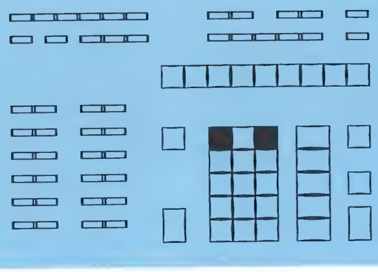
EXAMPLE:

Find the number, x, whose hyperbolic tangent is 0.5

$$x = \tanh^{-1}(0.5)$$

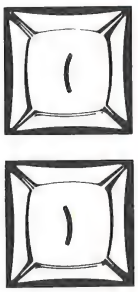
PRESS CLEAR . 5 *hyper* *arc* or *arc* *hyper* *tan x*

The result is .5493061443



KEYBOARD

PARENTHESES



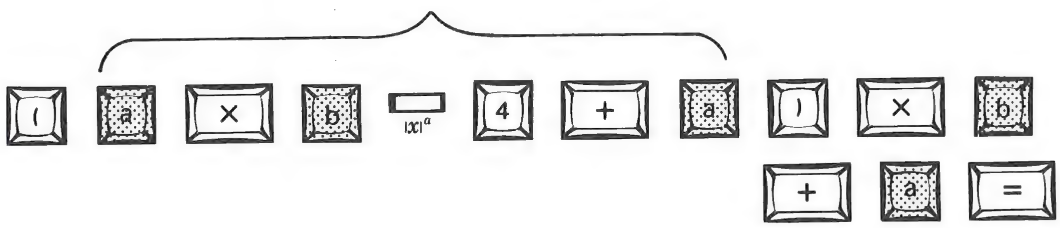
The open paranthesis key, (, and the close parenthesis key,), are used to organize complex mathematical statements.

If we enclose a series of terms and operators in parentheses, the calculator will execute the enclosed terms before it executes the non-enclosed terms. To illustrate, consider the equation below.

$(a \cdot b^4 + a) \cdot b + a = y$ (a and b are data entries.)

To solve this equation on the calculator we would enter the following keystrokes.

ENCLOSED TERMS

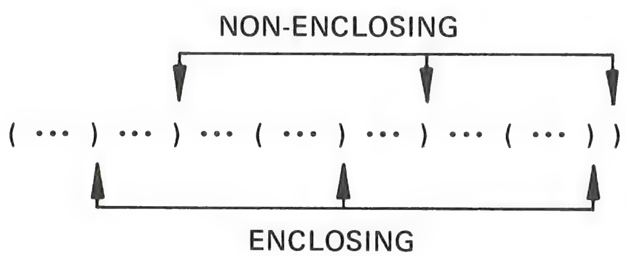


When the close parenthesis key is pressed, the calculator accumulates the result of the enclosed terms and puts it in the display. The equation is thus reduced to

Where c is the accumulated result of the enclosed terms.

USING THE PARENTHESES KEYS

In an expression a close parenthesis may be either of two types: enclosing or non-enclosing. A definition of each is contained below.

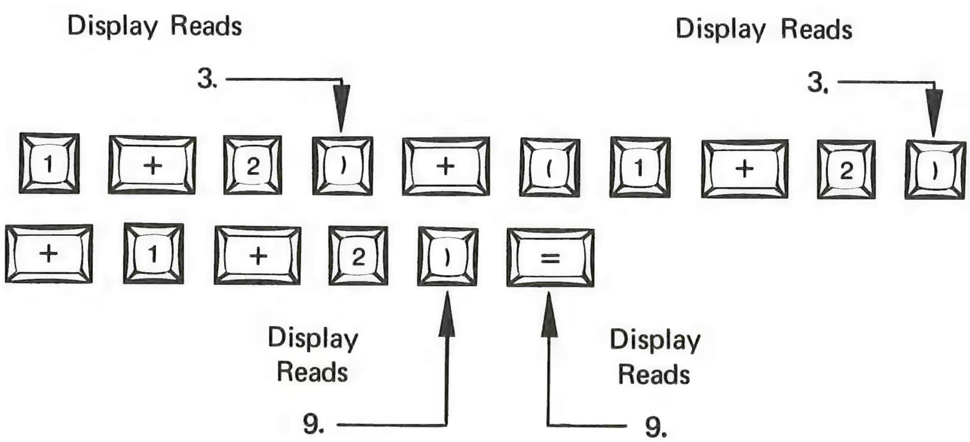


* An enclosing) calls for accumulation of the enclosed terms.

* A non-enclosing) calls for accumulation of all preceding operations.

EXAMPLE:

If you perform the following keying sequence, you will find that each time the close parenthesis key is used, the result of preceding operations, as explained above, will appear in the display.



PARENTHESES

The calculator will not recognize more than one successive (in an expression.

EXAMPLES:

These statements are valid	This statement is *not* valid
2 X (2 + 5) = (2) X (2 + 5) =	(2 X (2 + 5)) =
In each of the above, the correct result, 14, will be obtained.	In the above, the second (is not recognized by the calculator and the result will be 9, not 14.

From the above you may correctly deduce that it is not valid to nest parentheses but you may chain parenthetical expressions.

Not valid:

··· (··· (···) ···) ···

Valid:

··· (···) ··· (···) ···
···) ···) ···) ···) ···
···) ··· (···) ···) ···

EXAMPLES:

Below are a few expressions that you might encounter while solving problems. Study the keying sequences opposite the expressions to understand the techniques involved in the use of parentheses.

1. $(2 \times 2) + 3$ $\boxed{2}\ \boxed{\times}\ \boxed{2}\ \boxed{+}\ \boxed{3}\ \boxed{=}$

2. $2 \times (2 + 3)$ $\boxed{2}\ \boxed{\times}\ \boxed{(}\ \boxed{2}\ \boxed{+}\ \boxed{3}\ \boxed{)}\ \boxed{=}$

   or

   $\boxed{2}\ \boxed{\times}\ \boxed{(}\ \boxed{2}\ \boxed{+}\ \boxed{3}\ \boxed{=}$

3. $\sqrt{3^2 + (4 + 5)^2}$ $\boxed{3}\ \boxed{}_{\sqrt{\Sigma x^2}}\ \boxed{(}\ \boxed{4}\ \boxed{+}\ \boxed{5}\ \boxed{)}\ \boxed{=}$

   or

   $\boxed{3}\ \boxed{}_{\sqrt{\Sigma x^2}}\ \boxed{(}\ \boxed{4}\ \boxed{+}\ \boxed{5}\ \boxed{=}$

4. $\dfrac{25}{(3 + 2)}$ $\boxed{2}\ \boxed{5}\ \boxed{\div}\ \boxed{(}\ \boxed{3}\ \boxed{+}\ \boxed{2}\ \boxed{)}\ \boxed{=}$

   or

   $\boxed{2}\ \boxed{5}\ \boxed{\div}\ \boxed{(}\ \boxed{3}\ \boxed{+}\ \boxed{2}\ \boxed{=}$

5. $\dfrac{25}{(3 + 2)} + 7$

   $\boxed{2}\ \boxed{5}\ \boxed{\div}\ \boxed{(}\ \boxed{3}\ \boxed{+}\ \boxed{2}\ \boxed{)}\ \boxed{+}\ \boxed{7}\ \boxed{=}$

Following the above guidelines is generally simple and straight-forward. Usually only the most complex statements need simplification. When simplification is called for the data registers may be used to store intermediate results which can be recalled later to calculate a final result.

KEYBOARD

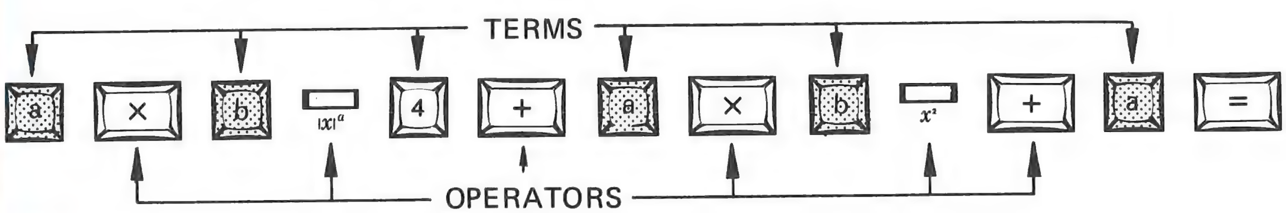
HIERARCHY

Hierarchy refers to the priority or precedence of the mathematical operators. Consider the equation, Y = A + B × C. When verbally expressing the relationships between A, B, and C, care must be exercised to rule out misinterpretations. For example, does the expression mean, "A plus the quantity B times C" or does it mean "C times the quantity A plus B? These expressions are different and only one is correct. The point to be made here is that the calculator must interpret relationships of this kind, i.e., a hierarchy or priority of operations must be established in order to rule out misinterpretations. The calculator does this by using the hierarchy relationships that have been long established as a part of everyday mathematics.

A mathematical statement consists of *terms* separated by *operators*, the operators giving the relationship between the terms. Consider the statement below:

$$y = a \cdot b^4 + a \cdot b^2 + a \text{ (a and b are data entries)}$$

The statement would be written on the calculator as



Inspection of the above reveals that each term is separated by an operator. The calculator inspects the statement and accumulates the result of the operations according to the hierarchy of the operators.

1. The highest priority is assigned to those operators which immediately perform a complete operation on the term in the display. These operators are: the trigonometric and hyperbolic keys, the register arithmetic keys, x!, ln x , log x , int x , e^x, x^2 , $\sqrt{x}$, and 1/x. In the example, the x^2 key has the highest priority and is executed first. Hence the statement is reduced to

2. Operations involving $|x^a|$ and $\sqrt{\Sigma x^2}$ are done next. In the example the x^4 term receives second priority; the statement is reduced to

   $\boxed{a}$ $\boxed{\times}$ b^4 $\boxed{+}$ $\boxed{a}$ $\boxed{\times}$ b^2 $\boxed{+}$ $\boxed{a}$ $\boxed{=}$

3. Multiplication and division receive third priority. In the example, $a \cdot b^4$ and $a \cdot b^2$ are calculated; the statement is reduced to

   $a \cdot b^4$ $\boxed{+}$ $a \cdot b^2$ $\boxed{+}$ $\boxed{a}$ $\boxed{=}$

4. Addition and subtraction have lowest priority. In the example, the terms $a \cdot b^4$, $a \cdot b^2$ and a are summed and the final result, y, is displayed. The statement has been reduced to its solution;

   $\boxed{=}$ DISPLAY

The calculator always follows the above sequences when accumulating the result of several operations. The numerical examples below illustrate these relationships.

EXAMPLE 1:

In a statement containing multiple levels of hierarchy, operators with the highest priority receive the first treatment.

$\boxed{1}$ $\boxed{+}$ $\boxed{2}$ $\boxed{\times}$ $\boxed{3}$ $\boxed{|x|^a}$ $\boxed{2}$ $\boxed{x^2}$ $\boxed{=}$ 163 (Result)

$2^2 = 4$

$3^4 = 81$

$2 \times 81 = 162$

$1 + 162 = 163$

EXAMPLE 2:

In a statement containing operators with equal levels of hierarchy, the operations are performed on a first-come-first-served basis.

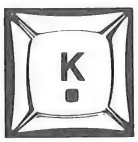
$\boxed{3}$ $\boxed{\sqrt{\Sigma x^2}}$ $\boxed{4}$ $\boxed{|x|^a}$ $\boxed{3}$ $\boxed{=}$ 125 (Result)

$\sqrt{3^2 + 4^2} = 5$

$5^3 = 125$

DATA REGISTERS

Data registers are those portions of the calculator memory that are devoted to data storage. The data registers are of two types: K-registers and R-registers. The K-registers differ superficially from the R-registers in that you will address and use them differently, but for data storage purposes they are the same.



There are ten K-registers, K_0 through K_9. Each of the K-registers has a single digit address. In addition to their data storage functions, the K-registers are related to the keys that perform register arithmetic. (These keys are discussed in the next chapter.)

In the basic calculator there are 64 R-registers. (With optional memories it is possible to expand to 1000 registers.) The R-registers have three digit addresses and are addressed either directly or indirectly. Data may be transferred from the R-registers to magnetic tape and vice-versa through a tape transport mechanism.

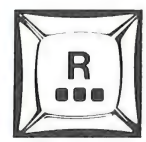
REGISTER ORGANIZATION

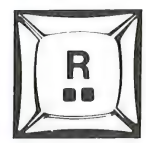
The ten K-registers are identified by single digit addresses, 0 through 9. The digits designate registers K_0 through K_9. K_0 through K_4 are related to the register arithmetic keys.

The R-registers are organized into files. Each complete file contains 100 R-registers. (On the basic calculator, the 64 R-registers partially fill the first file, file 0. Expanded memories allow for 1000 R-registers or 10 complete files.)

DATA REGISTERS

Every R-register can be identified by a three digit address, fdd. The first digit (f) specifies the file number; the remaining two (dd) specify the R-register in the file. Whenever any R-register is accessed by its three digit address, the calculator remembers the file number (f) of the accessed register. The calculator will continue to operate in this file and the registers in it may be subsequently accessed by their two digit addresses (dd) *until* the current file is changed by accessing an R-register in a *different* file with another three digit address. (The current file may be determined by entering the Learn mode. See OPERATING MODES.)





Notice that there are two Register addressing keys on the keyboard, R■■■ and R■■. The R■■■ key requires a 3 digit address and is used to access any register in any file. The R■■ key requires a two digit address and is used to access any register within the *current* file as mentioned in the preceding paragraph.

CLEAR
R FILE ■

There is one file addressing key on the keyboard. CLEAR R FILE. This key requires a one digit file address and is used to clear all R-registers within the addressed file.

Whenever any R or K key is depressed, the calculator expects that an appropriate number of digital addressing keystrokes will follow. This expectation is announced to you by illumination of the *Address Incomplete light*. If the address is not completed before a non-digit keystroke is entered, an error message (E 3) will appear in the display. Reset this error message by completing the address. If a non-existent R-register is addressed, an error message (E 2) will appear in the display. Reset this error message with a CLR. (See APPENDIX for further explanation of error messages.)

DATA REGISTERS

DIRECT STORAGE OPERATIONS WITH THE K AND R-REGISTERS

Pressing the equals key, =, prepares the calculator for a storage operation. Any register addressing sequence that follows an = will cause storage of the displayed number in the addressed register (K or R).

EXAMPLE: Store π in K_5 and $1/\pi$ in R_{49}.

1) PRESS [π] to enter π into the display.

2) PRESS [=] [K] [5]

3) PRESS [$\frac{1}{x}$]

4) PRESS [=] [R] [0] [4] [9]

RESULT: π is stored in K_5 and $1/\pi$ is stored in R_{049}.

To display and examine the contents of any storage register, simply press, in sequence, the keys that identify the register of interest. (Display R_{fdd} by pressing R■■■ f d d. Examine the contents of K_d by pressing K d.)

EXAMPLE: To recall the contents of the registers used in the previous example:

1) PRESS [K] [5] to recall the contents of K_5 (π).

2) PRESS [R] [4] [9] to recall the contents of R_{049} ($1/\pi$).

Note here that we did not use the full address of R_{049}. This was not necessary because we should be operating in file 0 from the previous example.

STORAGE AND THE PROGRAM CONTROL KEYS

When a displayed number is set for storage by an =, it remains set for storage through *all the following* keystrokes [RESET] [ADDRS] [GO TO ■■■■] [FROM TAPE ■] [TO TAPE ■] [STOP] [PRINT DSPLY] [PAPER FEED] [GO TO DISPLAY] [CLEAR FLAG] [SET FLAG] [CLEAR R FILE ■] [PAUSE] [FLAG] [FLASH] [<Ø] [=Ø] [≥Ø] [CONT]

To illustrate,

[=] [K] [0] causes storage of the display in the K_0 data register.

in addition,

any combination of the above keystrokes and [=] [K] [0] also causes storage of the display in the K_0 data register

This again prompts the suggestion that the = be used with care, and preferably only for storage operations.

UTILIZING DIRECT STORAGE AND RECALL OPERATIONS IN PROBLEM SOLUTIONS

Up until now, during each calculation you performed on the calculator, you have been required to key-in each data entry as it was needed in the course of a solution. In addition, you have been required to manually record or print each significant result. By incorporating the added flexibilities of the storage registers, a significant portion of the efforts involved in problem solutions are reduced and you thus gain another degree of freedom in the methods by which problem solutions may be implemented.

To illustrate this point, consider the form of the equations below:

$$y = Mx + B = K_0 \times K_1 + K_2$$

Obviously,

$$M = K_0$$
$$x = K_1$$
$$B = K_2$$

DATA REGISTERS

If we store the numerical values of M, x, and B in the storage registers K_0, K_1, and K_2, respectively, we may then solve the equation *in terms of* the contents of these registers and thus negate the necessity of keying-in the data entries as they are needed during the solution.

Suppose M = 2.571

x = 11.72

B = 5.713

Store the data in the appropriate storage registers. (Here we will use the K-registers, but the R-registers could be used with equal ease.)

PRESS [CLEAR DSPLY] [2] [·] [5] [7] [1] [=] [K] [0]

[CLEAR DSPLY] [1] [1] [·] [7] [2] [=] [K] [1]

[CLEAR DSPLY] [5] [·] [7] [1] [3] [=] [K] [2]

Now proceed with the solution.

PRESS

[K] [0] [×] [K] [1] [+] [K] [2] [=]

(The result is 35.84512)

You may wish to repeat the calculation with new variables and store the new results.

First, store the old result in K_3.

PRESS [=] [K] [3]

Now suppose we wish to evaluate the equation for x = 3.7; store the new x in K_4.

PRESS [CLEAR DSPLY] [3] [·] [7] [=] [K] [4]

Now repeat the solution and store the new result in K_5 .

PRESS

[K] [0] [×] [K] [4] [+] [K] [2] [=] [K] [5]

(The result, which is stored in K_5, is 15.2257)

We may now recall all of the information from the storage registers.

PRESS [K] [0]	Display reads:	2.571	M
[K] [1]	Display reads:	11.72	x (first one)
[K] [2]	Display reads:	5.713	B
[K] [3]	Display reads:	35.84512	y (x =11.72)
[K] [4]	Display reads:	3.7	x (last one)
[K] [5]	Display reads:	15.2257	y (x =3.7)

In a more complex expression you may want to reduce the expression to several less complex expressions, store the intermediate results in the storage register, then accumulate these intermediate results to obtain the total result. For example the equation

$$y = \frac{ae^x + b}{2c + d} - (a \cdot \sin x + 3)^{(\tan x/c)}$$

might be evaluated as follows:

$ae^x + b$	R_{001}	
$2c + d$	R_{002}	Intermediate
$a \cdot \sin x + 3$	R_{003}	Results
$\tan x/c$	R_{004}	

R■■■ 0 0 1 ÷ R■■■ 0 0 2	Final
− R■■■ 0 0 3 x^a R■■■ 0 0 4 =	Result

DATA REGISTERS

Here, you have at your fingertips the ability to assign labels to the variables in a problem and recall them into the display at will.

Later you will find that the calculator not only has this ability, but also possesses the ability to store and remember *keystrokes*. This is programming, which will be covered later in the manual.

INDIRECT ADDRESSING

Indirect addressing is a method of addressing one R-register with the contents of a second R-register. The number stored in the second becomes the address of the first.

The indirect addressing sequence is similar in form to the direct addressing sequence except that there is one extra R keystroke. Later in this section we will discuss the relationships between the two R keys in indirect addressing but for now we will stick to the generalities of the technique using *only* the R key, R■■■. The indirect addressing sequence is:

R■■■ R■■■ f d d where fdd is a complete R-register address.

If nnn are the 3 least significant digits of a number stored in R_{fdd}, then the sequence above will cause nnn to be taken as the address of the first R■■■ in the sequence.

R_{nnn} is $R_{R_{fdd}}$ when nnn are the 3 least significant digits of the number stored in R_{fdd}

DATA REGISTERS

The storage and recall sequences are similar to the sequences used in direct addressing. To store, precede the addressing sequence with an equals keystroke, and to recall, simply use the addressing sequence alone.

STORAGE: = R■■■ R■■■ f d d

RECALL: R■■■ R■■■ f d d

To illustrate indirect addressing, consider the following operations:

1) Store the number 17 in register R_{033}.

PRESS: [CLEAR DSPLY] [1] [7] [=] [R] [0] [3] [3]

2) Now store π indirectly in register R_{017}.

PRESS: [ ] π [=] [R] [R] [0] [3] [3]

3) The result of the above is that the calculator will inspect the 3 least significant digits of the contents of R_{033} and find that they are 017 which are then used as the address for the first R in the sequence. Thus, π is stored in R_{017}.

4) To check that the above is true, recall π from R_{017}; this may be done either directly or indirectly. Try it both ways.

Direct Recall, PRESS [R] [0] [1] [7]

Indirect Recall, PRESS: [R] [R] [0] [3] [3]

DATA REGISTERS

When using indirect addressing, *any* number may be stored in the address register, but only the *last three* digits (least significant) will be used as the indirect address.

EXAMPLE:

Store the number 1024 in R_{001},

PRESS:

[CLEAR DSPLY] [1] [0] [2] [4] [=] [R] [0] [0] [1]

Now store π indirectly through R_{001}.

PRESS: [π] [=] [R] [R] [0] [0] [1]

Result: Find π in R_{024}.

On the basic calculator (64 R-registers), if a number greater than 064 had been stored in R_{001}, then the calculator would have found that it did not possess the addressed register and an error message, (E 2), indicating *no such register*, would result in the display. The same applies with other memory options: An error message will result whenever a non-existent register is addressed.

RELATIONS OF THE R–REGISTER KEYS IN INDIRECT ADDRESSING

Either of the two R keys, R■■■ or R■■ may be used in the indirect addressing sequence. The results will vary with the order and sequence used.

Remember that the first digit (f) of the three digit address (fdd) that follows the R■■■ keystroke will change the operating file number. With this in mind, realize that care must be exercised when mixing the indirect addressing keystrokes in order that you do not inadvertently change the operating file and proceed to call R-registers from the wrong file with the short form addressing sequence, R■■ R■■ d d.

✱ If the second keystroke is R■■■ as in

R■■ R■■■
or f d d
R■■■ R■■■

then the address in R_{fdd} becomes the address for the first R in the sequence and the *file is changed* to coincide with the just-entered f keystroke.

✱ If the second keystroke is R■■ as in

R■■ R■■
or d d
R■■■ R■■

then the address in register dd of the *current* file becomes the address for the first R in the sequence and the file number *remains unchanged*.

REGISTER ARITHMETIC

The term 'register arithmetic' loosely groups the set of keys that simplify some of the addition, multiplication and counting operations commonly used in programming. Each of these keys manipulates specific combinations of stored or displayed data, altering the contents of one of the five storage registers, K_0 through K_4, in the manner indicated on the keys. The individual keys perform the following operations:

Key	Operation	Effect
Σ_0	Display + K_0 = K_0	(display unchanged)
Σ_1	Display + K_1 = K_1	(display unchanged)
${}_3\Sigma_2$	$K_3 + K_2 = K_2$	(K_3 and display unchanged)
Π_4	Display X K_4 = K_4	(display unchanged)
Δ_3	(–0.1) X K_3 = K_3	(display unchanged)

These keys do not prevent the use of the storage registers for other purposes, do not change the display, and do not affect the hierarchy of a statement. They perform without affecting any calculation in progress.

Σ_0

Σo Adds the displayed data to the current contents of register K_0 and stores the sum in K_0. The display remains unchanged.

Σ_1

Σ1 Same as Σo except it uses the K_1 register.

EXAMPLE:

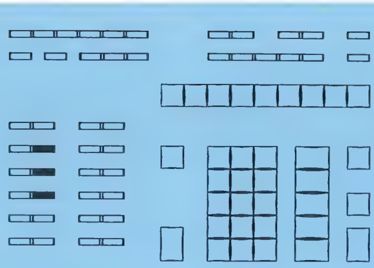
Twenty eight is a 'perfect' number; all of its integer factors add to 2 X 28. Use the storage register K_0 to sum the factors 1, 2, 4, 7, 14, and 28.

PRESS	Display	K_0
[CLEAR DSPLY] [=] [K] [0]	0000000000	0
[CLEAR DSPLY] [1] [Σ]	0000000001	1
[CLEAR DSPLY] [2] [Σ]	0000000002	3
[CLEAR DSPLY] [4] [Σ]	0000000004	7
[CLEAR DSPLY] [7] [Σ]	0000000007	14
[CLEAR DSPLY] [1] [4] [Σ]	0000000014	28
[CLEAR DSPLY] [2] [8] [Σ]	0000000028	56

Now recall contents of K_0

PRESS	Display	K_0
[K] [0]	0000000056	56

(The next perfect number is 496.)



KEYBOARD

REGISTER ARITHMETIC

Π_4

$\Pi 4$ multiplies the displayed data by the current contents of register K_4 and stores the product in K_4. The display remains unchanged.

EXAMPLE:

Obtain the product of a series of entries

		Display	K_4
PRESS	[CLEAR DSPLY] [1] [=] [K] [4]	0000000001	1
	[CLEAR DSPLY] [2] [Π_4]	0000000002	2
	[CLEAR DSPLY] [3] [Π_4]	0000000003	6

To recall the product

PRESS	[K] [4]	0000000006	6

${}_3\Sigma_2$

${}_3\Sigma 2$ adds the current contents of K_3 to the current contents of K_2. The contents of K_3 are unchanged and the display is not affected.

EXAMPLE:

Store 0 in K_2 and 5 in K_3. Sum the contents of K_3 in K_2 six times.

PRESS

	K_2	K_3
[CLEAR DSPLY] [=] [K] [2] [5] [=] [K] [3]	0	5
[${}_3\Sigma_2$]	5	5
[${}_3\Sigma_2$] [${}_3\Sigma_2$]	15	5
[${}_3\Sigma_2$] [${}_3\Sigma_2$] [${}_3\Sigma_2$]	30	5

PRESS [K] [2] Is the result 30?

[ ] Δ_3

Δ_3 multiplies the current contents of register K_3 by -0.1 and stores the result in K_3. The display is unaffected.

EXAMPLE:

PRESS	K_3
[CLEAR DSPLY] [1] [0] [0] [0] [=] [K] [3]	1000
[ ] Δ_3	-100
[ ] Δ_3	$+10$
[ ] Δ_3	-1
[ ] Δ_3	$+.1$

This key is useful in finding the roots of equations through iterative zero-crossing search techniques.

PERIPHERAL CONTROL

REMOTE, RMT, sets the calculator to control a remote peripheral device.

This key is used to send data to or receive data from remote instruments.

The RMT key can be used as a direct address key, or can be used with a K-register for indirect addressing. Each peripheral to be used with the calculator is assigned a two-digit identification number.

To directly address a given peripheral

PRESS [REMOTE] [d] [d]

where the two digit keys represent the address of the peripheral.

To indirectly address a peripheral, the address digits must be the two least-significant digits of the mantissa stored in a given register R_{dd}

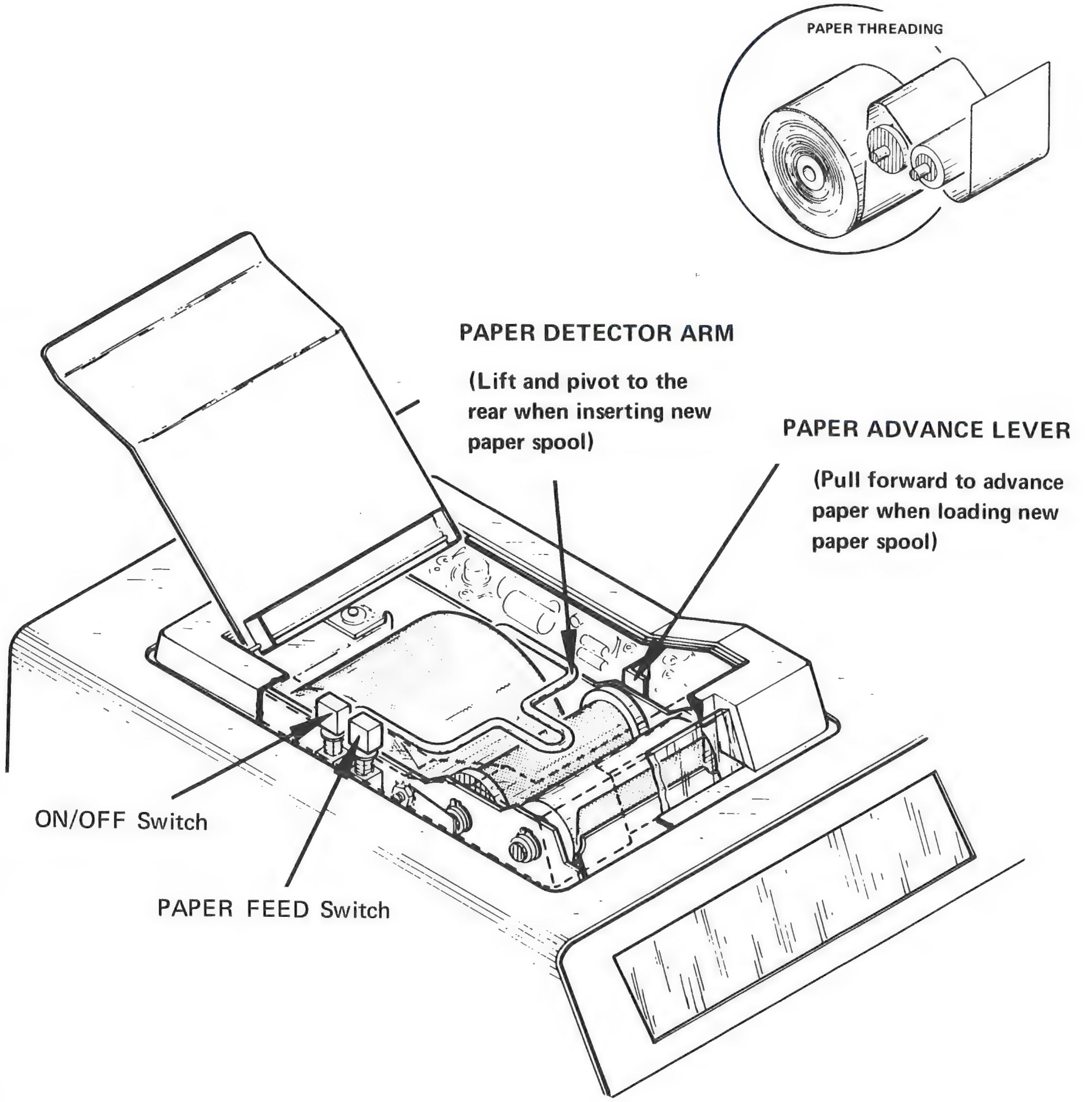
PRESS [REMOTE] [R] [d] [d]

and the calculator will look in R_{dd} find the address digits, and access the designated peripheral.

Details of peripheral operations are given in the peripheral manuals.

PERIPHERAL CONNECTOR

PRINTER



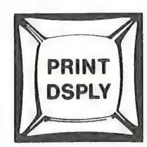
The paper detector arm (shown in the above illustration) will place the calculator in the Busy mode if the printer runs out of paper. If this should occur either turn off the printer or install a new roll of paper. DO NOT ATTEMPT TO DEFEAT THE PAPER DETECTOR. Serious damage to the thermal print head and/or roller may result if the paper detector mechanism is defeated.

PRINTER

To turn the printer on, depress the printer ON-OFF switch located on the printer mechanism on top of the calculator.

To advance the paper, depress the PAPER FEED switch located in front of the printer ON-OFF switch.

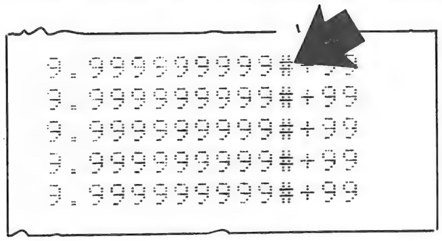
The remaining printer control keys are a part of the basic calculator keyboard and are included on all calculators, whether or not a printer is actually installed. If your calculator was not originally equipped with the silent thermal printer, one may be installed at any time without keyboard replacement. The printer control keys generate their respective program codes independently of the printer itself, hence programs with print-control commands may be written, executed, and recorded on calculators not equipped with printers.



The print key, PRNT, commands the internal printer to print the contents of the display, and then advance the paper by one line, without changing the display.

* All numbers are printed as displayed, except that non-significant leading 0's are suppressed. Displays consisting of all zeros are printed as "0".

* If the display is flashing when printed, a pound sign will appear in the printout, as illustrated:



PAPER FEED

The paper feed key, PF, advances the paper one line each time it is depressed.

PROGRAMMING

INTRODUCTION TO PROGRAMMING

Until now, our discussion has centered upon the use of the calculator as a sophisticated adding machine; by now you know how the various keys on the lower half of the keyboard operate relative to the display and to each other. You also know, from our discussion on Storage Operations, that the calculator has some sort of memory — at least so far as data is concerned. But this is only the half of it; remember, the *calculator is programmable*. This means, quite simply, that the calculator has the ability to remember, in its program memory, *all* that you have become familiar with in the preceding text. Heretofore we have looked at the keys on the lower half of the keyboard; now we will discuss the remainder, the programming keys. But first, let's look at programming in general.

What is a program? A program is a series of instructions (keystrokes) that, when stored in the calculator program memory, may be executed by the calculator in a continuous manner, from start to finish. The instructions that make up a program are of two kinds. You are already familiar with the first kind of instructions, these are the keys on the lower half of the keyboard — you have used them before, and they need no further explanation except that they are programmable. The second kind of instructions are those associated with the keys on the upper half of the keyboard. Generally, these keys relate to program *flow*, directing the order and manner of execution of program steps that may be physically separated from each other in memory, but are linked by virtue of program organization.

Let's look at some sample programs in order to clarify the above. The simplest kind of program is one that contains no branches. That is, the instructions in the program are located adjacent to one another in memory and are executed sequentially by the calculator, one after another. As an example, enter a program that will take a number from the display, store it in K_0, and stop.

INTRODUCTION TO PROGRAMMING

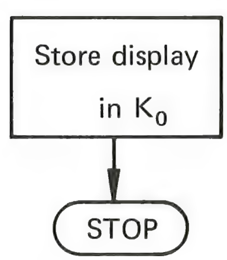
PRESS	
RESET	Set the calculator at the beginning of its memory, location 0000.
LEARN	Enter the Learn mode, in which the calculator will remember keystrokes, at 0000.
= K 0	These keystrokes — as you already know — cause the number in the display to be stored in the K_0 data register.
CLEAR DSPLY	Clear the display.
STOP	This will be the last program step, an instruction to the calculator telling it to STOP program execution.
LEARN	Exit the Learn mode; we are done programming.

The program and its flowchart look like this:

PROGRAM

memory locations	stored keystrokes
0000	=
0001	K
0002	0
0003	CD
0004	STOP

FLOWCHART



To run the program, enter a number into the display and press START. The START causes the calculator to execute the instructions in its memory, *starting* at location 0000. Thus, the display is stored in K_0, cleared, and the program stops — note that the *STOP* light is on. Recall the contents of K_0 to verify storage (press K 0).

INTRODUCTION TO PROGRAMMING

Now let's enter another program into the calculator memory. Have this second program recall K_0, add one, and then STOP with the sum displayed (K_0 + 1). Locate the new program beginning at location 0100 in memory.

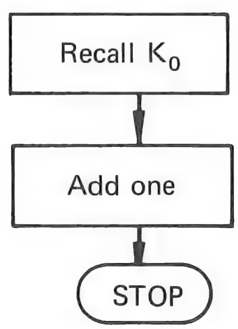
PRESS	GO TO 0 1 0 0	Set the calculator to memory location 0100.
	LEARN	Enter the Learn mode.
	K 0	Recall K_0 into the display.
	+ 1)	Add one to K_0.
	STOP	Stop and display the sum.
	LEARN	Exit the Learn mode.

The program and flowchart look like this:

PROGRAM

0100	K
0101	0
0102	+
0103	1
0104	)
0105	STOP

FLOWCHART



To execute the program press GO TO 0 1 0 0 and CONT. As directed, the calculator will recall K_0, add one, and STOP with the sum in the display.

INTRODUCTION TO PROGRAMMING

Now let's link the two program segments together. To do this, enter the Learn mode at location 0004, the location of the STOP in the first program. Then enter the program instructions: GO TO 0 1 0 0 to direct program flow to the beginning of the second program segment. After this, exit the Learn mode.

PRESS

GO TO 0 0 0 4	Set the calculator to the memory location of the STOP in the first program.
LEARN	Enter the Learn mode.
GO TO 0 1 0 0	These instructions, which overwrite the previous ones, direct the calculator to go to 0100, the location of the second program.
LEARN	We are done; exit the Learn mode.

As before, to run the program enter a number into the display and press START. As a result, the number in the display will be stored in K_0, incremented by one, and displayed.

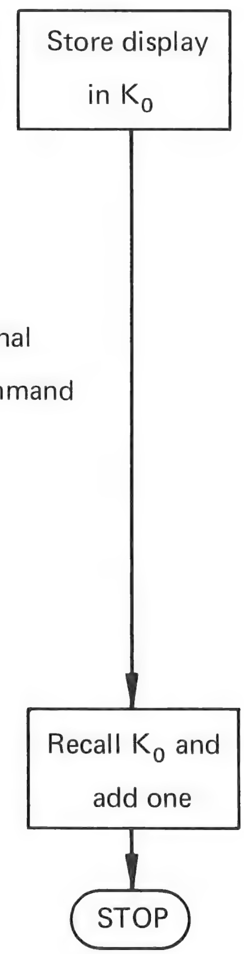
INTRODUCTION TO PROGRAMMING

Our program and flowchart now look like this:

PROGRAM

0000	=	
0001	K	
0002	0	
0003	CD	
0004	GO TO	This is an unconditional branch command
0005	0	
0006	1	
0007	0	
0008	0	
0009		ignored
⋮		
0099		
0100	K	
0101	0	
0102	+	
0103	1	
0104	)	
0105	STOP	

FLOWCHART



INTRODUCTION TO PROGRAMMING

Now look again at the flowchart and program. Notice that the program steps, GO TO 0 1 0 0 are not shown in the flowchart, GO TO and the address digits following it are only intended to achieve continuity in program execution and are not necessary in the program flowchart. This is true of most of the remaining keys on the upper half of the keyboard; they control the program and tell the calculator information about the program. All of the computation in this or any other program is done with the keys on the lower half of the keyboard.

Notice that in our simple program example we have used RESET, LEARN, STOP, START, and GO TO, five of the 31 keys on the upper half of the keyboard. You already understand what these keys do in a program and you will find the rest of the keys equally simple to understand. Programming is really quite simple — just a series of instructions, executed one at a time. You are at step five of 31. Now get with the program! Turn the page to execute the next step.

PROGRAM MEMORY

The portion of memory devoted to the storage of program steps is called the program memory. On the basic calculator without additional memory options, the program memory consists of 512 program steps. Each step is capable of storing a single keystroke. With expanded memory, as many as 8192 program steps are available.

MEMORY ORGANIZATION

Program steps in memory are organized into *pages*. Each page contains 1000 program steps. On the basic calculator there is only one page, page 0, partially filled with 512 steps. There are as many as eight full pages in the expanded memory.

PROGRAM STEP COUNTER

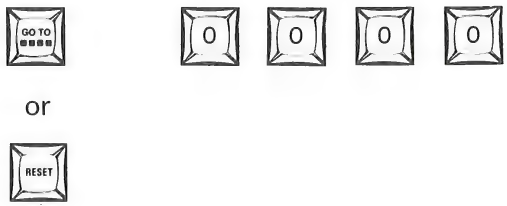
Sequential entry or execution of program steps is controlled by means of the program step counter. At any given time, the program step counter points to a single memory location. The step counter is automatically advanced one step when a program step is either entered or executed at the indicated location.

ENTERING AND STORING A PROGRAM

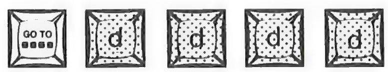
There are two methods of entering a program into the calculator program memory. The first method involves the entire keyboard; ; the second involves only a few keys and the magnetic tape cartridge. Here we will discuss program entry from the keyboard. For information on tape entries please refer to the section entitled **MAG TAPE**

To enter a program from the keyboard, first direct the calculator to the location in memory where you wish to begin storing the program steps. To start programming at step 0000, direct the calculator to the 0000 location with either of the two following keying sequences:

PROGRAM MEMORY



To start programming at any other location, the sequence



(d = any digit)

will direct the calculator to step dddd of the program memory.

Once the starting address is properly located, enter the Learn mode by pressing the green learn key, LRN. When the Learn mode is entered, the display will change form to show the program step currently indexed by the program step counter, an octal keycode representing the keystroke presently stored at that memory location, and a single digit showing the last accessed R-register file number.

dddd ddd d ← Last addressed R-register file number

Octal keycode of stored keystroke

Presently indexed program step number

Entry into the Learn mode is also indicated by illumination of the *learn* light that is located under the display.

As program steps are keyed in, the program step counter advances one step at a time, and the display will always show the keystroke presently stored at the indexed location. After the program is entered, exit from the Learn mode is accomplished by pressing LRN a second time. The *learn* light will extinguish and the program step counter will automatically be set to step 0000. The calculator will then wait there, in the Idle mode, for further instructions.

OPERATING MODES

Current machine status or *Operating mode* is indicated by illumination of one or more of the status lights located below the display.

RAD	DEG	LEARN	BUSY	STOP	ADDR INCOMP

In the preceding keyboard sections we have already discussed DEG and RAD, two of the six status lights. The remaining four convey program entry and operation information, informing you whether the calculator is currently *learning* a program, busy *executing* a program, has *stopped* program execution, or is *waiting* for an address.

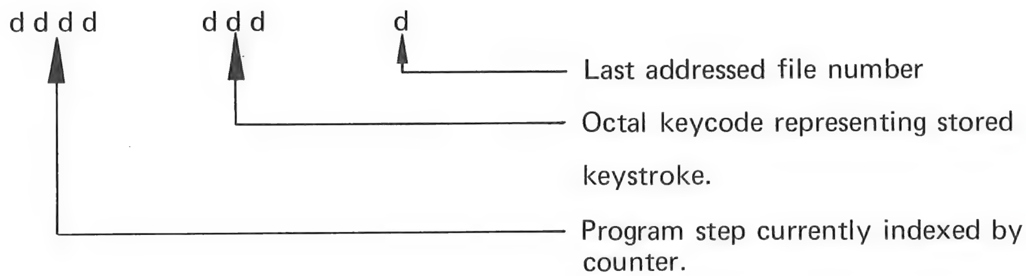
IDLE MODE

In the Idle mode: the *learn* and *busy* lights are extinguished and the calculator may be directed into any other mode.

LEARN MODE

The Learn mode is activated by pressing the green learn key, LRN, and is indicated by illumination of the *learn* light under the display. In the Learn mode, the programmable memory is opened to accept and retain sequential entry of all keystrokes except LRN and the editing keys. (The editing keys are located in the row immediately below the display.)

In the Learn mode the display consists of three number segments. The first segment of the display shows the four digit address of the program step currently indexed by the program step counter. The second segment gives an octal keycode representing the keystroke presently stored at this memory location. The last segment of the display consists of a single digit which indicates the file number of the last addressed R-register.



Certain of the editing keys can only be used in the Learn mode.

They are:

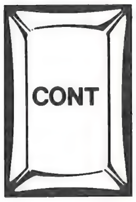
Exit from the Learn mode is accomplished by pressing LRN a second time. Exit from the Learn mode is always into the Idle mode.

BUSY MODE

The Busy mode is indicated by illumination of the *busy* light under the display. The calculator is put into the Busy mode whenever it is actually processing information. This occurs momentarily after every keystroke and is most noticeable during lengthy operator computations. (Try 6 9 x!). The Busy mode is also entered when the calculator is executing a program, transferring information to/from tape, and when it is performing certain editing functions. Busy mode also occurs whenever the calculator is under control of a remote peripheral device.

OPERATING MODES

A programmed RSET (and under certain conditions, an END) encountered by the calculator during program execution will cause exit from the Busy mode. In addition, a programmed or manual STOP command will also cause exit. After a STOP is encountered the program step counter remains at the location following the encountered STOP.



Resumption of the interrupted program is accomplished by pressing CONT. An encountered RSET (and under certain conditions, an END) will set the program step counter to step 0000.

Exit from the Busy mode is always into the Idle or stop mode.

NOTE

Except for the STOP key, the keyboard is disabled during the Busy mode. To abort any program being executed, press the STOP key and then reset the calculator by pressing the RESET key.

STOP MODE

The Stop mode is indicated by illumination of the *stop* light under the display. Entry into the Stop mode is accomplished whenever the calculator encounters a programmed STOP command. A programmed STOP will cause the calculator to halt program execution at the location following the STOP.

To re-enter the Busy mode and resume program execution press CONT. This will cause the interrupted program to resume execution at the step after the encountered STOP command.

ADDRESS INCOMPLETE MODE

This mode is indicated by illumination of the *address incomplete* light under the display. Entry into this mode follows all encountered keystrokes that require a subsequent address. These keys are:

GO TO ■■■■	Requires a four digit address (dddd)
R ■■■	Requires a three digit address (ddd)
R ■■	Requires a two digit address (dd)
REMOTE ■■	Requires a two digit address (dd).
CLEAR R FILE ■	Requires a one digit address (d)
TO TAPE ■	Requires a one digit address (d)
FROM TAPE ■	Requires a one digit address (d)
K ■	Requires a one digit address (d)
EXECUTE ■	Requires a one key address (d)
LABEL ■	Requires a one key address (d)

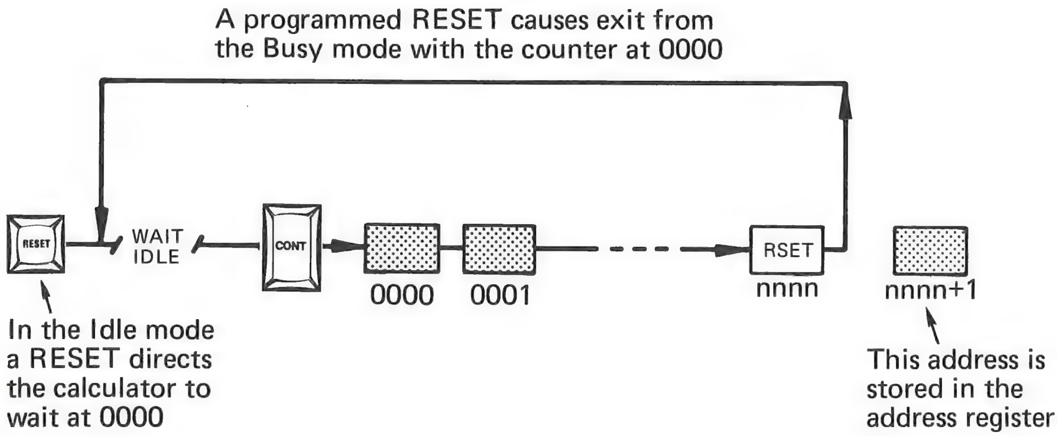
Exit from the Address Incomplete mode is accomplished by satisfying the required address. If the proper address is not entered before pressing another key, an error code (E 3) is displayed along with the step, keycode, and file.

PROGRAM CONTROL KEYS

The program control keys are used to control the execution of your program. When placed in your program, they direct the calculator to branch, stop, make a decision pause, or terminate the program execution.

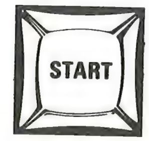
RESET

In the Idle mode, RSET directs the program step counter to step 0000. The calculator will wait in the Idle mode for further instructions. If you desire to start execution of the program steps starting at 0000, follow the RSET command with a START keystroke or CONTINUE keystroke.

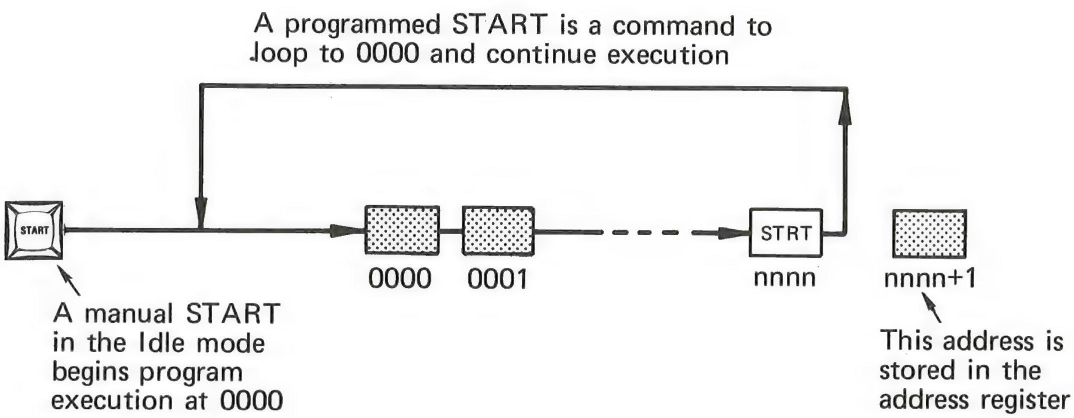


In the Busy mode, when the calculator encounters a stored RSET command, it reacts by directing itself to step 0000. The calculator will then exit the Busy mode, enter the Idle mode, and wait for further instructions.

START

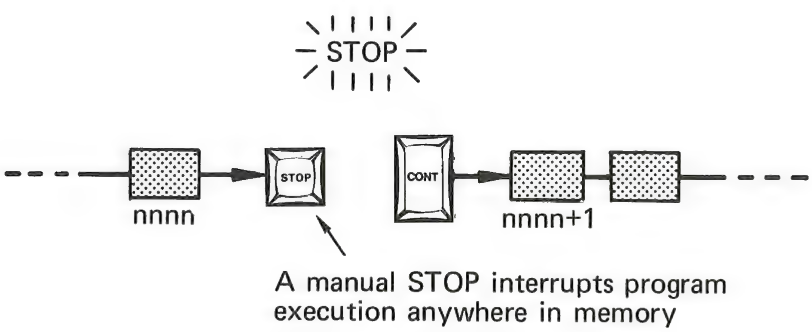


In both the Idle and Busy modes, a STRT will direct the calculator to step 0000 and begin program execution.



STOP

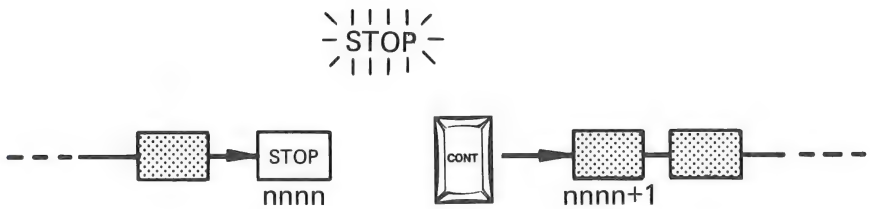
The STOP command halts program execution. It is the only key that remains functional in the Busy mode, and may be used at any time to manually halt program execution.



If a program execution is interrupted by a manual STOP command, the calculator will stop at the step following the step that was being executed when the manual STOP was entered; to resume, press CONT, or to terminate the program, press RESET.

PROGRAM CONTROL KEYS

In the Busy mode, whenever the calculator encounters a programmed STOP, program execution will halt at the location following the STOP and enter the Stop mode with the stop light illuminated. The program step counter will contain the address of the location following the STOP.



A programmed STOP halts execution at the location following the STOP command.

In the Stop mode the keyboard is fully functional.

≥Ø

=Ø

<Ø

IF CONDITION

The condition part of the IF CONDITION term refers to the state of either the display or flag. When the calculator encounters a programmed IF keystroke, it examines the display or flag and queries them about their existing condition. The query has a yes or no answer; a yes applies when the CONDITION exists, and a no applies when the CONDITION does not exist.

PROGRAM CONTROL KEYS

KEYSTROKE	CONDITION	YES	NO
≥Ø	Is the display greater than or equal to zero?	The display is greater than or equal to zero.	The display is negative.
=Ø	Is the display equal to zero?	The display is equal to zero.	The display is positive or negative.
<Ø	Is the display negative?	The display is negative.	The display is positive or zero.
FLASH	Is the display flashing?	The display is flashing.	The display is not flashing.
FLAG	Is the FLAG set?	The FLAG is set.	The FLAG is not set.

FLASH

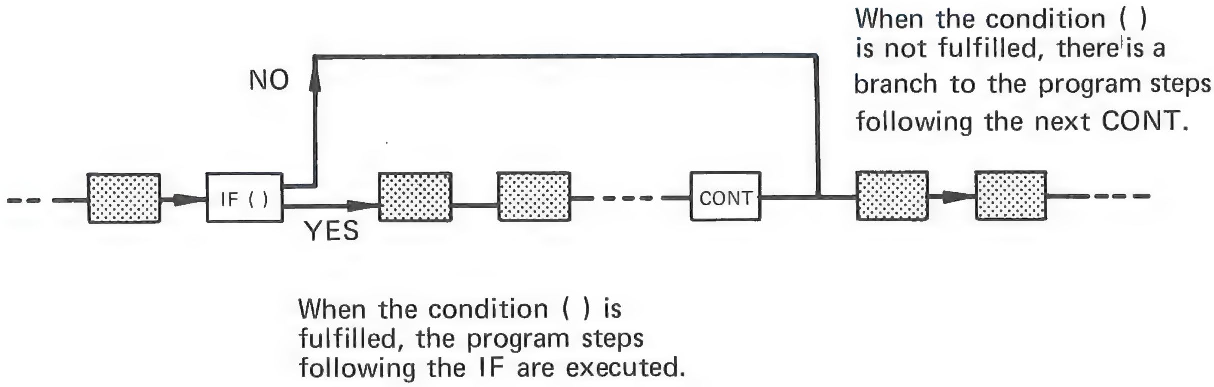
FLAG

SET FLAG

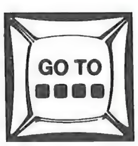
CLEAR FLAG

The FLAG is an internal latch that is *set* by means of the SET FLAG key, SFG, which is located above and to the right of the IF CONDITION keys. The flag is *cleared* by pressing CLEAR FLAG or RESET.

PROGRAM CONTROL KEYS



If the calculator finds, after inspection of either the display or the flag, that the stated CONDITION is true (yes), the calculator will execute the program steps immediately following the IF statement. If the CONDITION is found to be false (no), the calculator will branch to and execute the program steps following the next stored CONT keystroke.

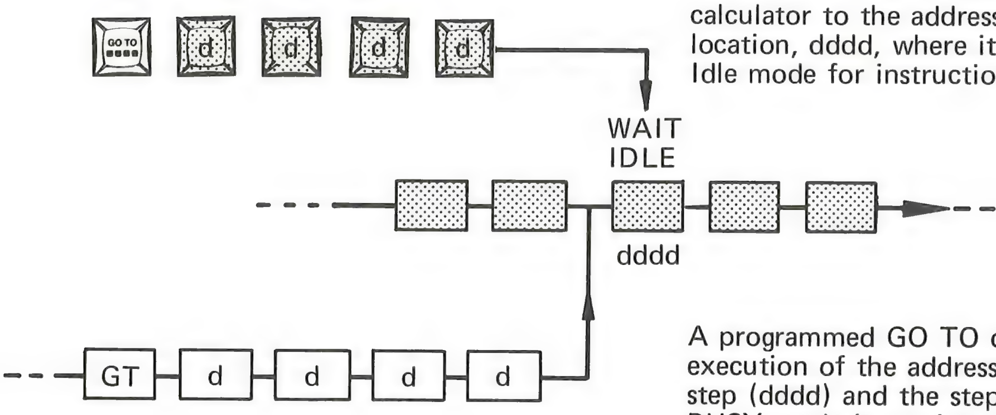


GO TO

The GO TO command, GT, must be followed by a four digit address, dddd, which will extinguish the Address Incomplete light. If GT is followed by a non-existent address, error code (E 1) will result. If GT is followed by an incomplete address (less than 4 digits) error code (E 3) will result.

In the Idle mode, GT d d d d will cause the calculator to branch to program step dddd and wait there in the Idle mode for further instructions.

PROGRAM CONTROL KEYS



A manual GO TO command directs the calculator to the addressed memory location, dddd, where it will wait in the Idle mode for instructions

A programmed GO TO command causes execution of the addressed program step (dddd) and the steps following it. BUSY mode is not interrupted.

In the Busy mode, whenever the calculator encounters the programmed sequence, GT d d d d , the calculator branches to step dddd and execution of the stored program continues at that location. GT d d d d is considered to be an unconditional branch command. (For a further discussion of branching, please refer to the section entitled **PROGRAMMING HINTS.**)

STEP →

The STEP FORWARD key, STP→ is non-programmable and is used to advance the program step counter one step at a time in the Learn and Idle modes.

In the Busy mode, STP→ is non-operable.

In the Idle mode, STP→ is used to sequentially execute stored program steps. To accomplish execution of program steps stored at any location, simply direct the calculator to the program steps of interest with a GT d d d d and press STP→ sequentially. Further discussion of the STP→ key is contained in the section entitled **PROGRAM EDITING AND DEBUGGING.**

PROGRAM CONTROL KEYS

GO TO DISPLAY and RETURN ADDRESS

Whenever the calculator encounters a GO TO DISPLAY keystroke, GODP, the calculator will branch to the program step number given by the four least-significant digits in the display. If the display does not contain a valid address, an error code, (E 1), will appear instead.

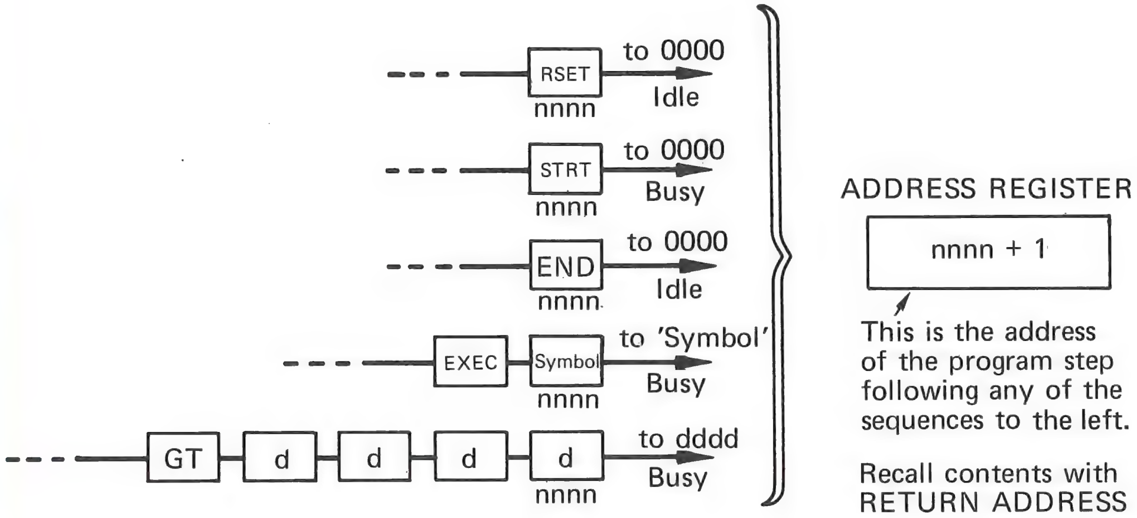
In the Idle mode, GODP will cause the calculator to branch to the displayed step number.

In the Busy mode, GODP will cause the calculator to branch to the displayed step and execution of the program will continue from that location.

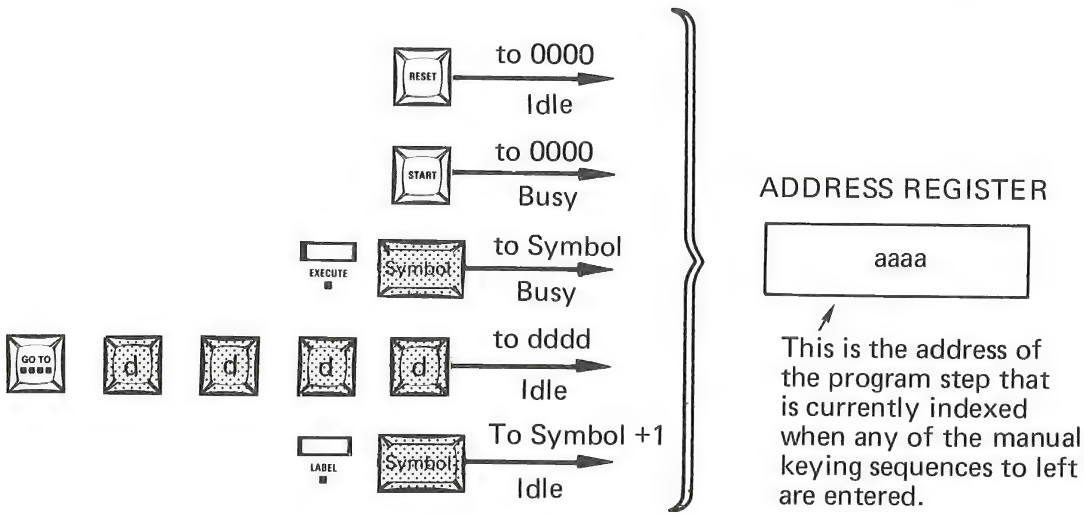
GO TO DISPLAY

RETURN ADDRESS

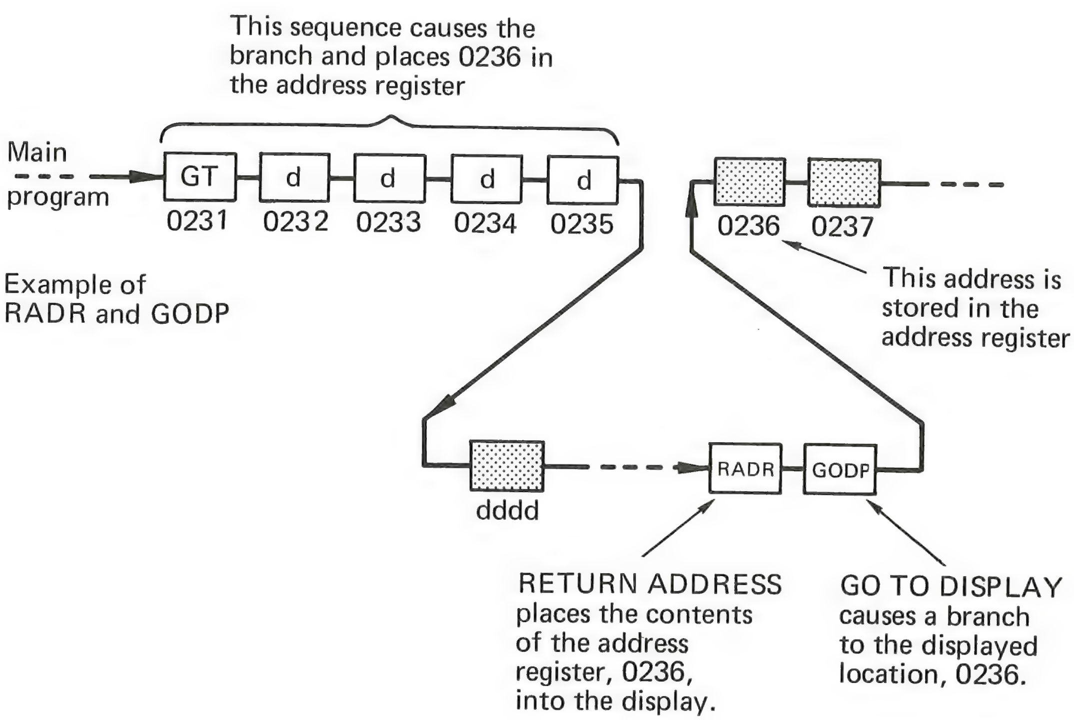
The GO TO DISPLAY key is normally used in conjunction with the RETURN ADDRESS key, RADR. When the calculator encounters a RADR keystroke, it recalls to the display an address that was previously stored in a special address register. The stored address is the address of the program step that follows any of these sequences:



PROGRAM CONTROL KEYS



The RADR keystroke works the same in both the Busy and Idle modes.



CAUTION: RADR overwrites the current contents of the display. For further explanation of the Return Address key, see the section on **SUBROUTINES**.

PROGRAM CONTROL KEYS

PAUSE

LABEL

HOLD FOR ALPHA

EXECUTE

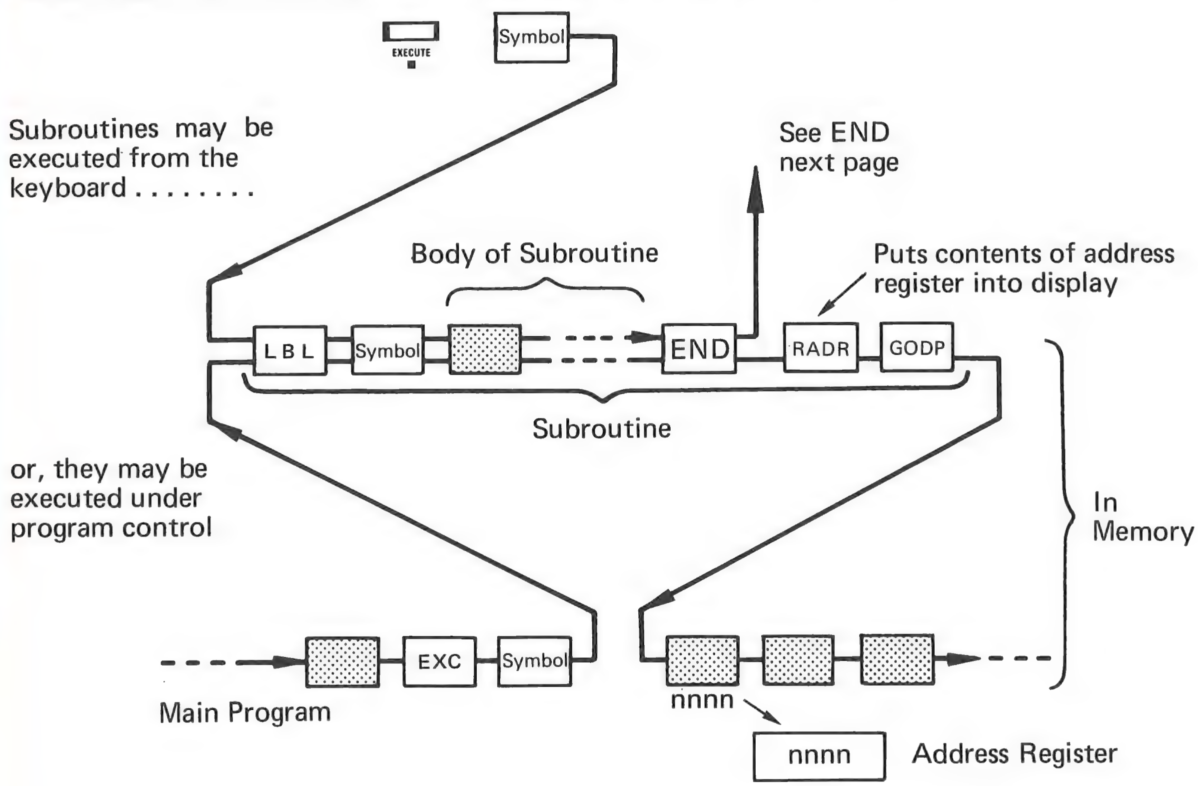
PAUSE

In both the Idle and Busy modes, the programmable PAUSE keystroke, PAUS, is used to delay the program execution for approximately one second. Sequential pauses cause correspondingly longer delays. Program one or more PAUS when you wish to observe the result of a programmed calculation in the display without stopping program execution.

Subroutines programmed into the calculator may use the LABEL key, LBL, in the sequence LABEL *SYMBOL* where *SYMBOL* is the labeling keystroke. *SYMBOL* may be any one of 151 keys (including alpha keys). The LABEL key, however, should be avoided as a *SYMBOL*.

ALPHA symbols may be obtained by pressing the HOLD FOR ALPHA key while pressing any of the alpha keys. The alpha keys are the parentheses, numeric, and those designated by adjacent blue squares.

The EXECUTE keystroke, EXC, is always followed by another keystroke, *SYMBOL*, which informs the calculator which (labeled) subroutine to execute.



PROGRAM CONTROL KEYS

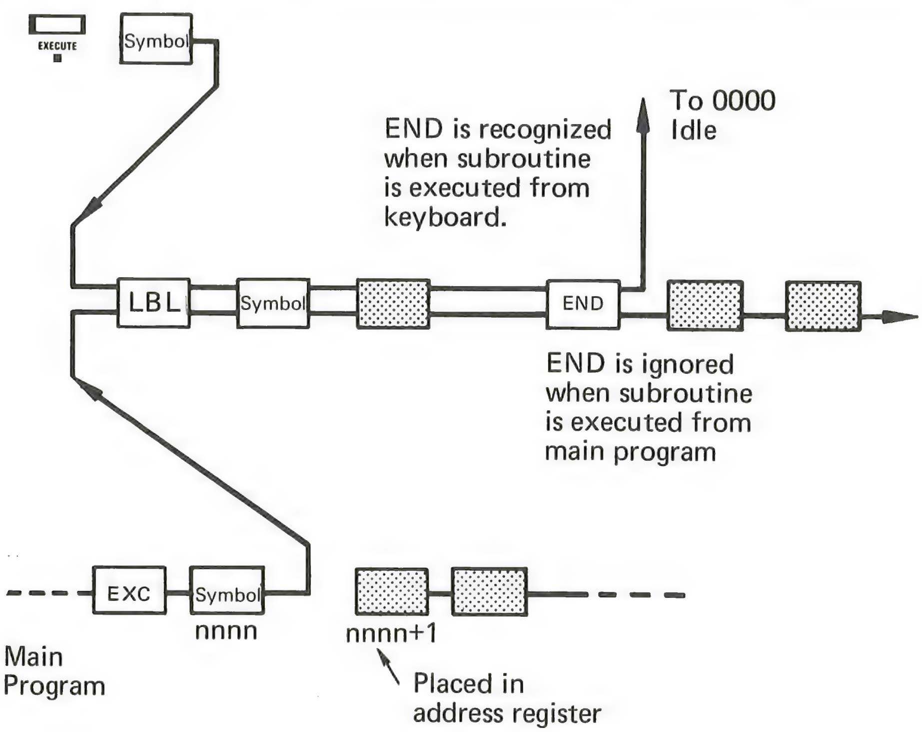
In the Idle mode, the sequence EXC *SYMBOL* (where *SYMBOL* is a label) will cause the calculator to branch to the beginning of the subroutine that is labeled *SYMBOL*. After finding the subroutine, execution of the program steps contained in it will proceed until an END is encountered or until a key code is encountered that will take it out of the Busy mode.

In the Busy mode, the programmed sequence EXC *SYMBOL* will cause the calculator to branch to and execute the program steps contained in the subroutine labeled *SYMBOL*. Note that any END contained in the subroutine will be ignored by the calculator when the subroutine is executed as a result of a programmed EXC *SYMBOL* command.

END

END

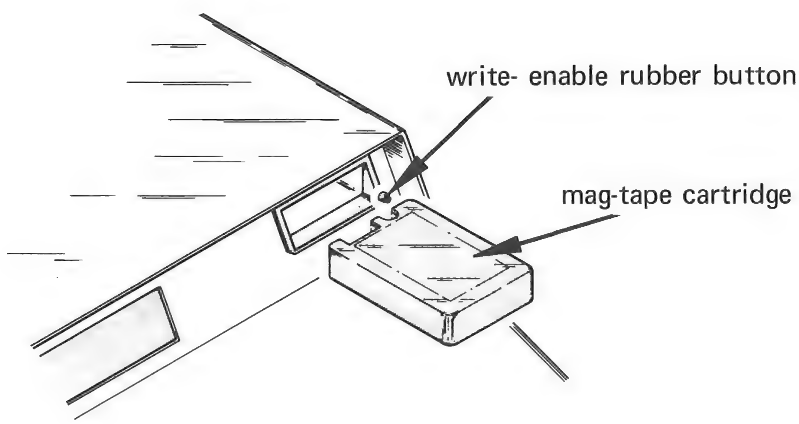
The END key is used in subroutines. When a subroutine containing an END is executed from the keyboard by EXC *SYMBOL*, the END causes the calcuator to branch to step 0000 and END execution. The first END stop encountered terminates execution even if it is a subroutine that is called by the original subroutine. However, when a subroutine containing an END is executed as a result of a command such as STRT or CONT, the END is ignored.



MAG TAPE

The calculator has a *volatile* program memory. That is, the contents of the calculator's programmable memory are erased whenever AC power to the calculator is interrupted. However, it is possible to compose a permanent record of stored programs and data, that is a recording placed through the magnetic tape transport mechanism onto a magnetic tape. On the mag-tape we can *record* both R-register data and program steps. The process is simple. Equally simple is the *loading* process, in which the contents of a prerecorded tape are loaded into the calculator memory. In addition, using a combination of recording and loading, programs and data files may be conveniently altered at will; this is *editing*.

The mag-tape cartridge contains an endless loop of high quality digital magnetic tape. The tape is divided into six segments, called *blocks*, that are separated from each other by a series of punched-out holes. The holes in the tape allow optical identification of the various blocks by means of a photo-transistor mounted in the transport mechanism.



On the front of the cartridge is a hole designed to accept a rubber button. When the tape cartridge, with rubber button in place, is inserted into the transport mechanism, the button depresses a microswitch which allows you to record on the tape. When the cartridge is write-protected (rubber button removed), no information can be recorded on the tape, it can only be read. In order to record information onto the tape the rubber button must be inserted. If you attempt to write on a write-protected cartridge, an error message (E 8) will appear in the display.

Both program steps and R-register data may be recorded on the mag-tape. Remember that a *page* consists of 1000 program steps and a *file* consists of 100 R-registers. Each of the six tape blocks will accept a full page, a full file, or portions thereof; *however, page and file information (program steps and data) may not be mixed on a block of tape.*

When information is transferred from the calculator memory *to* a tape block, the information previously recorded on the tape block is replaced by the new information. The information stored in the calculator memory is not altered by the transfer.

When information is transferred *from* a tape block to specified locations in the calculator memory, the current information contained in the calculator memory is replaced by that coming in from the tape block. The information on the tape block is not altered in this process.

When either page or file contents are transferred from the calculator to a tape block, the calculator recognizes the end-of-page (program step d999) or the end-of-file (R-register f99). When either is encountered, even if a full page or file has not been recorded, the rest of the tape block is erased. Thus only page or file contents up to their respective boundaries (d999 or f99) are recorded. This will be extremely useful when you are later manipulating (editing) program steps and data with the mag-tape.

MAG TAPE

RECORDING PROGRAM STEPS OR DATA ON TAPE

To record program steps or R-register data on a tape block, we must first tell the calculator where in its memory to begin the recording. The address key, ADR, is used for this purpose.

When transferring program steps, the ADR keystroke is followed by GO TO and a four digit addressing sequence, dddd, that identifies the *location* in memory where the recording is to begin. To transfer file information, follow the ADR keystroke with R■■■ or R■■ (whichever is appropriate) and their respective digits to indicate the starting R-register.

Once the beginning program step or R-register has been specified as above, a TO TAPE keystroke, TTP, informs the calculator that we wish to transfer the previously specified information *to* a tape block. A final keystroke, b, indicating the tape block (0 – 5) that is to receive the new information will initiate the transfer process.

The complete sequences look like this:

ADDRS GT d d d d TTP b	Transfers program steps, starting at dddd and ending at the end-of-page, to tape block b.
ADDRS R■■ d d TTP b	Transfers R-register data in *current* file, starting at dd and ending at the end-of-file, to tape block b.
ADDRS R■■■ f d d TTP b	Transfers R-register data in file f, starting at dd and ending at the end-of-file, to tape block b.

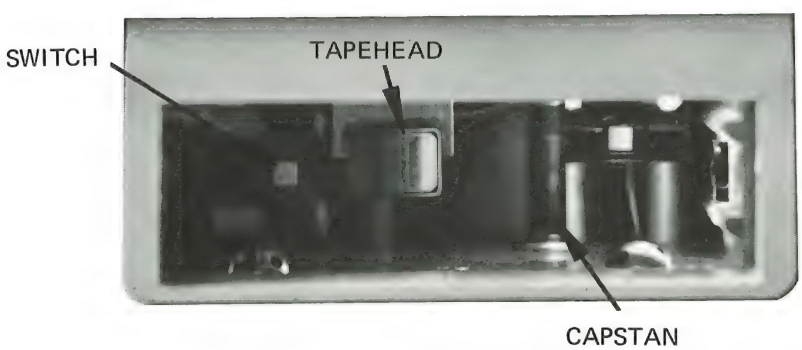
The transfer process begins as the calculator enters the Busy mode and searches the tape for the specified tape block, b. During this time a STOP will interrupt the process. Once the specified block has been found, information from the calculator, up to the end-of-page or file, is recorded on the tape block. During this period a STOP has no effect. When the transfer is complete, the calculator exits the Busy mode and re-enters the Idle mode.

An error message (E 7) will result whenever either the tape cartridge is not fully inserted into the transport mechanism, or when a non-existent tape block number (b greater than 5) is called for. An error message (E 8) results whenever the sequence, TO TAPE b, is used and the cartridge is write-protected (rubber button removed).

Error message (E 9) results whenever a *bit error* occurs during tape transfer. A bit error is defined as more than eight bits in a character. Since only eight bits may be stored in one location in memory or tape, more than eight bits will result in some sort of an error. A bit error may occur under a variety of anomalous circumstances but is typically due to either a power line glitch or dirt on the tape head. When (E 9) occurs, repeat the transfer.

CLEANING TAPE HEAD AND CAPSTAN

Use a cotton swab and denatured alcohol to clean the tape head and capstan. The capstan can be rotated by depressing the left-hand orange SWITCH with a pencil and pressing FROM TAPE 0 keys. Hold the cotton swab lightly against the capstan to remove dirt. When finished press STOP to stop the capstan from rotating.



MAG TAPE

LOADING PROGRAM STEPS OR DATA FROM TAPE INTO THE CALCULATOR

To load prerecorded program steps or data into the calculator memory, as before, we must first indicate the beginning of the memory locations which will receive the new information.

PRESS [ADDRS] [GO TO] d d d d for program steps

or

PRESS [ADDRS] [R] d d
or
[R] f d d
for data

Now press the FROM TAPE key, FTP. Follow this with a single digit keystroke, b, to indicate the tape block *from* which the information is to be taken. Thus the only difference between loading and recording is the use of the TO TAPE key in recording and the FROM TAPE key when loading.

The complete sequences look like this.

[ADDRS] [GO TO] d d d d [FROM TAPE] b	Loads contents of tape block b into the calculator memory starting at location dddd.
[ADDRS] [R] f d d [FROM TAPE] b	Loads the data contents of tape block b into file f, starting at register dd.
[ADDRS] [R] d d [FROM TAPE] b	Loads the data contents of tape block b into the *current* file, starting at register dd.

After the b keystroke, the transfer process is initiated as indicated by illumination of the *BUSY* light. As before, during the time that the calculator is searching for the specified tape block, a manual STOP will interrupt the transfer process. After the block has been found and actual information transfer commences, a STOP will no longer halt the process. When the transfer is complete, the calculator exits the Busy mode, enters the Idle mode, and is again ready to execute further manual or programmed instructions.

Again, error messages (E 7) or (E 9), as discussed previously, will result when their respective error conditions occur; (E 7) when a non-existent tape block is addressed or when the cartridge is not fully inserted into the transport mechanism, and (E 9) when a bit error occurs during the transfer.

EXAMPLE:

Starting at 0000, enter the following program into the calculator memory.

PRESS	RESET LEARN	Enter the Learn mode at 0000.
	R 0 0 0 + 1 = R 0 0	Add one to R_{00}, store sum in R_{00}.
	= R R 0 0	Store R_{00} in $R_{R_{00}}$.
	START	To 0000 and repeat program.
	LEARN	Exit the Learn mode.

MAG TAPE

To run the program initialize R_{000} to zero (CD = R■■■ 0 0 0) and press START. The program will place the address digits of each R-register in each R-register (1 in R_{01}, 2 in R_{02}, etc.). When all the R-registers have been filled, the calculator will stop program execution and display error message (E 2), indicating that a non-existent register has been addressed. This means that all available registers have been filled.

Now let's make a recording of the program. Place the recording on tape block zero.

1. Insert the write-enable button into the front of the cartridge.

2. Place the cartridge firmly into the transport mechanism.

3. To record the program, press

RESET		Same as ADR GT 0 0 0 0
TO TAPE	0	Instruction to record on block zero

4. When the *BUSY* light extinguishes, the transfer is complete; tape block zero now contains a complete recording of the program.

You may wish to check the contents of the tape to verify that we have a complete and accurate record of the program. To do this, let's first alter the original program by inserting a STOP at 0000.

PRESS	RESET	LEARN	Enter the Learn mode at 0000.
	INSERT	STOP	Insert a STOP at step 0000.
	LEARN		Exit the Learn mode.

As the program now stands, the calculator will exit the Busy mode and enter the stop mode whenever the STOP at step 0000 is encountered. Try it after re-initializing R_{000} to zero.

Now load the pre-recorded program into the calculator, starting at step 0000.

PRESS [RESET] — Same as ADR GT 0 0 0 0

[FROM TAPE] [0]

When the *BUSY* light extinguishes, signaling completion of the loading process, press CD = R■■■ 0 0 0 to initialize the program. Then press START. You should get the same results as before. This verifies that the recorded program is correct.

Now let's record the R-register data that we have generated in the program. Store the record of file zero (R_{000} – R_{099}) on tape block one.

PRESS [ADDRS] [R ■■■] [0] [0] [0] — Record is to begin at R_{000}

[TO TAPE] [1] — Record above on block number one.

When the *BUSY* light extinguishes the transfer is complete.

To check that the record is accurate, let's first destroy the data in file zero.

PRESS [CLEAR] [CLEAR R FILE] [0] — Clear all data from file zero

MAG TAPE

Now load the data back into the file;

PRESS [ADDRS] [R] [0] [0] [0] — Specify beginning register.

[FROM TAPE] [1] — Load contents of block number one into the calculator R-register memory starting at the specified location.

TAPE TRANSFERS DURING PROGRAM EXECUTION

At times you may find it desirable to transfer data or program steps back and forth between the calculator and the various tape blocks *during* program execution. Since the tape commands are programmable, you may do this provided you understand the following items.

- Register manipulation is implemented in the same manner as discussed on the previous pages. You may program the sequence

  [ADDRS] [R] d d or [R] f d d [FROM TAPE] or [TO TAPE] b

  anywhere in a program to achieve file transfers.

- You may transfer program steps *to* a tape block by programming the sequence

  [ADDRS] [GO TO] d d d d [TO TAPE] b

  anywhere in a program. Remember that the transfer process will not affect the program steps stored in the calculator; therefore, the above sequence will not affect program operation.

* You may load program steps into the calculator *from* a tape block by programming the sequence

ADDRS GO TO ▪▪▪▪ d d d d FROM TAPE ▪ b

anywhere in the program. If the transferred program steps are loaded into the same page as the program steps currently being executed by the calculator, the program step counter is directed to step 0000, and execution of the program will proceed from there.

When any transfer is initiated under program control, as soon as the calculator recognizes the tape block digit, b, in the transfer sequence, the transfer begins and the calculator waits until the transfer is complete. During the time that the transfer is taking place, the program step counter is set to the program step following the b keystroke in the calling sequence. After the transfer is complete, program execution is resumed at the current location of the program step counter, *or*, as described above, when the new program steps are loaded into the same page as those just executed, the program will resume execution at step 0000. In any case, during the time that the transfer is taking place, the calculator remains in the Busy mode; the only indication that a tape transfer is going on is the slight sound produced by the transport mechanism.

SUBROUTINES

Subroutines are program segments that are subordinate to the main program. They are designed to be used by the main program during its execution. These program segments are self-contained *mini-programs*, and are executed under control of a *master* program.

This may seem complex, but it's really not when you realize that every time you use an arithmetic key in a program, you are actually calling for execution of a subroutine. Take $\sqrt{x}$, for instance. If you remember the longhand method of computing square roots, you know that the process is somewhat complicated. But when using the calculator, all you have to do is enter a number into the display, press $\sqrt{x}$ and the displayed number is immediately replaced by its square root. There is no magic here; when you press $\sqrt{x}$, the calculator does the longhand calculation for you by executing an internally stored program that results in the desired solution. Thus, all of the arithmetic keys are really subroutine execution commands.

The subroutine capabilities of the calculator allow you to write your own mini-programs that can be executed by the calculator as easily as those related to the various arithmetic keys.

Subroutines are important because they simplify programming. They are useful because they are self-contained and can be used from either the keyboard or any location in the main program.

SYMBOLIC LABELING

A *symbolically labeled* subroutine is a set of programmed instructions beginning with the key sequence LABEL and *SYMBOL*, where the *SYMBOL* is any programmable keystroke For example, a subroutine might be symoblically labeled LABEL 1, LABEL A, or LABEL x^2. The fact that a subroutine has the symbolic label x^2, does not mean that the subroutine squares a number; x^2 is just a name that has been arbitrarily assigned to the subroutine.

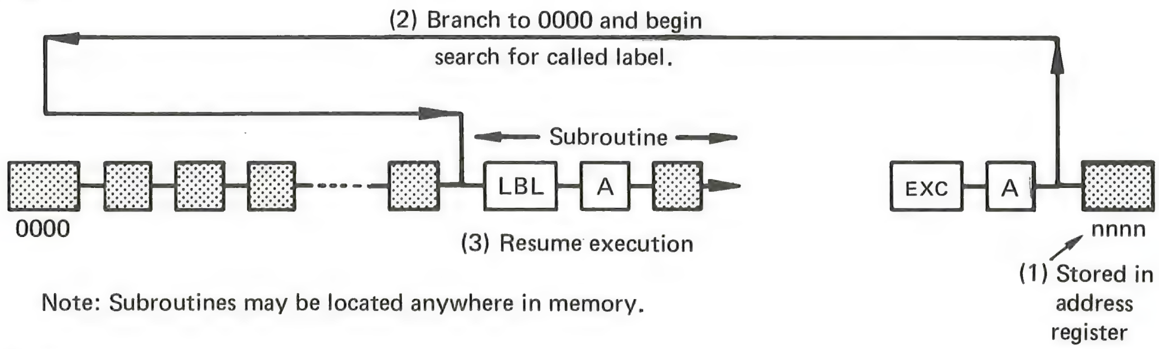
It is not necessary to assign a symbolic label to a subroutine. You might want to place a subroutine somewhere in memory without assigning it a name. Under these circumstances, you will have to remember where you placed the subroutine, and if you have several unlabeled subroutines in your program, keeping track of each subroutines location might be difficult.

Symbolic labeling of subroutines removes this burden. The calculator has the ability to search its memory and find labeled subroutines.

SUBROUTINE EXECUTION COMMANDS

A subroutine execution command is a series of keystrokes that tells the calculator to go to a certain portion of memory, then resume or begin program execution. This is often referred to as *calling a subroutine*. The keystrokes used to call a subroutine usually depend upon the type of subroutine being called, either labeled or unlabeled.

To call a symbolically labeled subroutine, use the calling sequence EXECUTE *SYMBOL*, where *SYMBOL* is the symbolic label (keystroke) assigned to the subroutine. For example, in EXECUTE 1, EXECUTE A, and EXECUTE x^2, the symbolic labels are 1, A, and x^2. The calling sequence causes the calculator to do two things: 1) the address immediately following the calling keystroke sequence is placed in the address register, and 2) the calculator goes to the beginning of its memory (location 0000) and begins a search for the labeled subroutine. When the calculator finds the subroutine with the label that is used in the calling sequence, it begins execution with the first program step after the label.

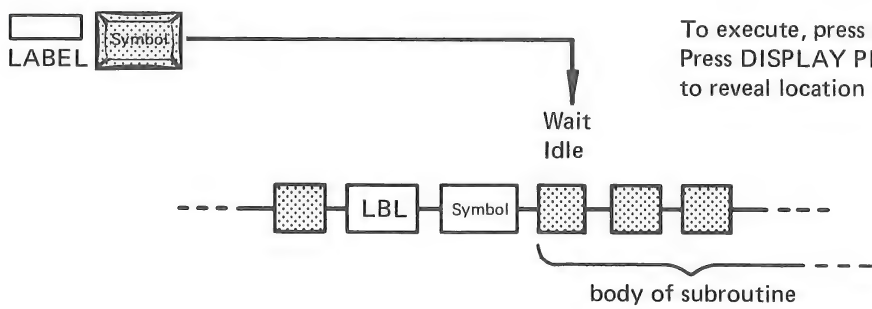


Note: Subroutines may be located anywhere in memory.

SUBROUTINES

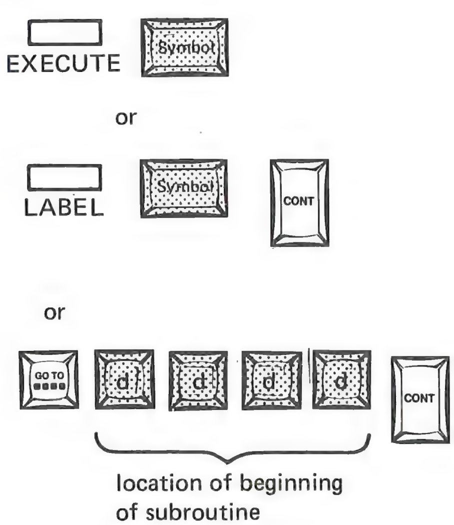
Labeled subroutines may be easily located for editing or debug purposes. When the calculator is in the Idle mode, the key sequence LABEL *SYMBOL* will set the program step counter to the program step immediately following the corresponding stored *SYMBOL* keystroke.

To locate a subroutine in memory, press

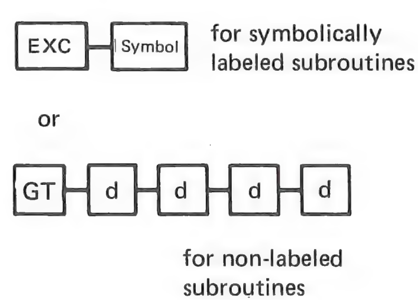


A subroutine, labeled or unlabeled, may be called using the GO TO key. The key sequence used is GO TO dddd where dddd is the first program step in the subroutine. As with the EXECUTE *SYMBOL* sequence, using GO TO dddd causes the calculator to do two things: 1) the address of the program step following the GO TO dddd sequence is placed in the address register and 2) the calculator program step counter is set to dddd. The calculator then resumes execution to step dddd.

To execute a subroutine from the keyboard in the Idle mode, press



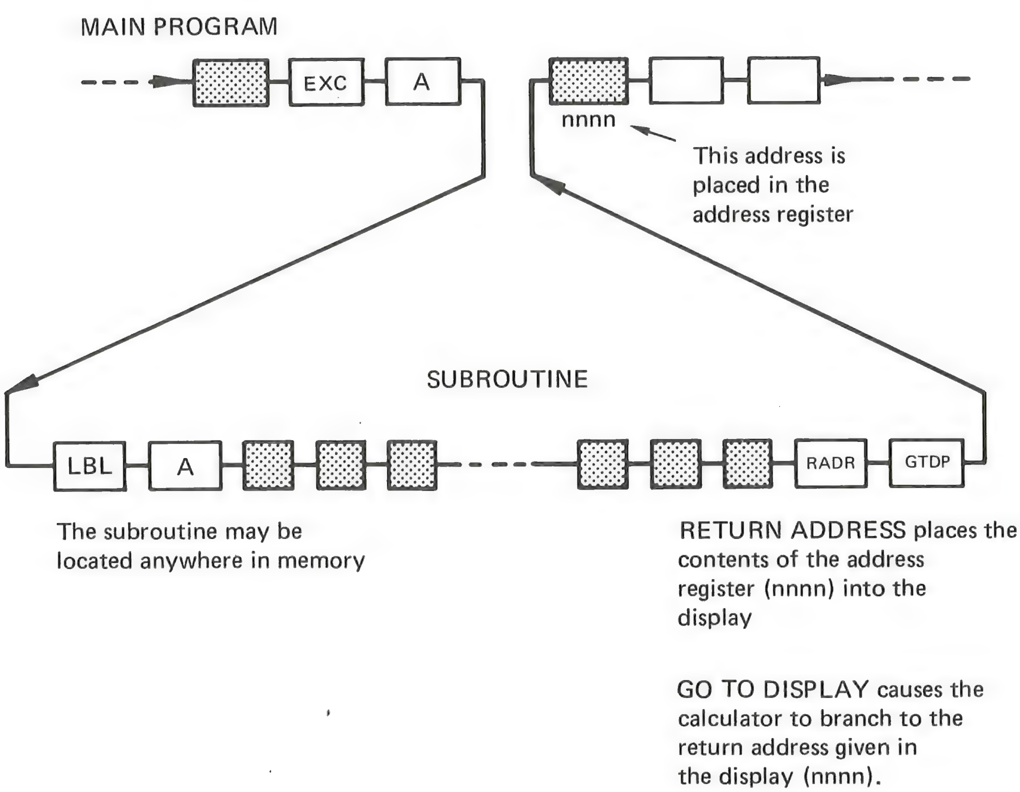
To execute a subroutine under program control, program these sequences



SUBROUTINE TERMINATIONS

After your program has called a subroutine and the program steps in the subroutine have been executed, the subroutine generally returns program control to the main program by setting the program step counter to the program step following the last keystroke in the calling sequence.

The simplest way to terminate your subroutines is with the keystrokes RETURN ADDRESS and GO TO DISPLAY. RETURN ADDRESS places the return address from the address register into the display. GO TO DISPLAY sets the program step counter to the displayed address and the calculator resumes program execution at the address.



SUBROUTINES

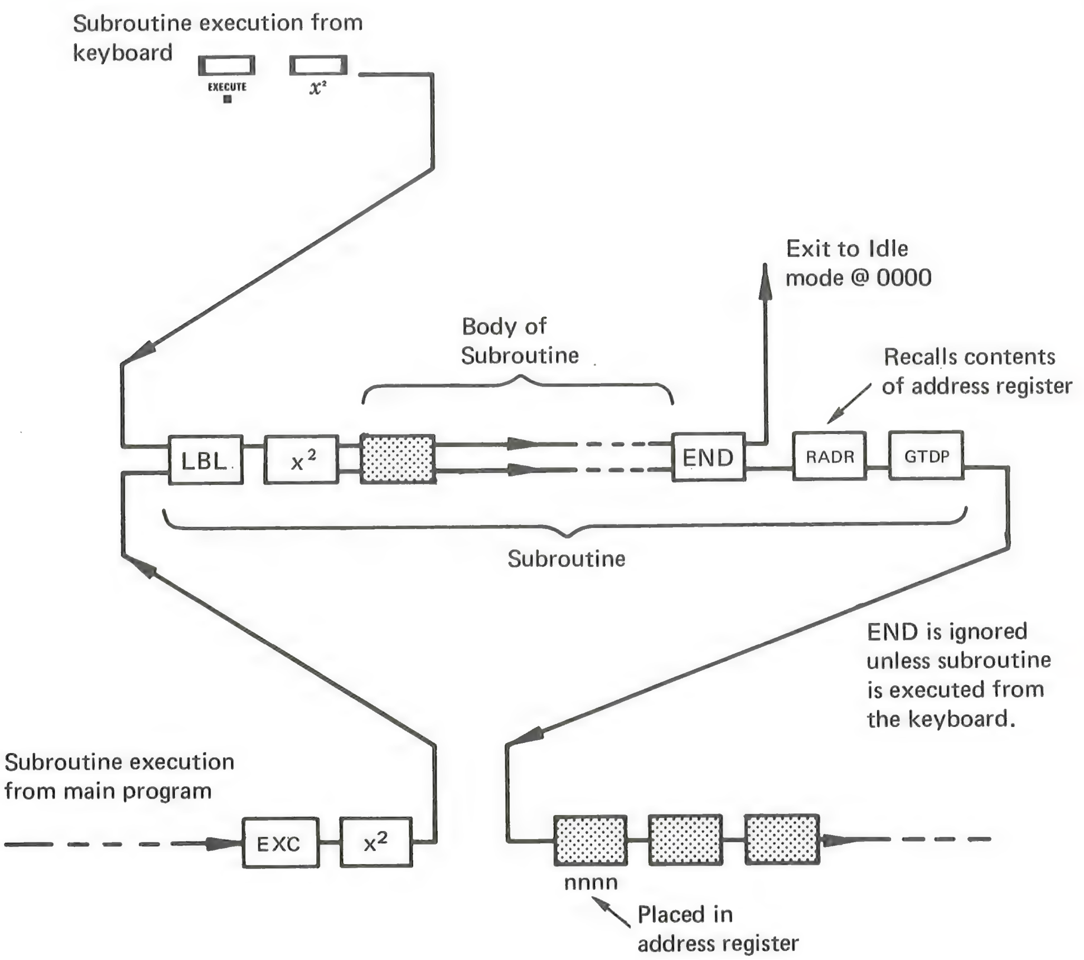
Thus, when you call a subroutine from some point in your program, the calculator automatically stores the return address in the address register. When the subroutine program steps are completed, the key sequence RETURN ADDRESS and GO TO DISPLAY automatically returns the calculator to the calling point. Notice that this capability allows you to call one subroutine from several points in your program and automatically return to the correct point every time.

The ease with which you can return from subroutines when using the RETURN ADDRESS, GO TO DISPLAY sequence, combined with the symbolic labeling of subroutines, makes it easy to write your programs by using subroutines as building blocks. The main program is then simplified to a series of EXECUTE *SYMBOL* keystrokes.

The preceeding discussion assumes that the subroutine was called by the program. Since a subroutine may be a mini-program that is complete within itself, you might want to execute just the subroutine without executing the entire program. For example, perhaps you have written a financial analysis program, which contains a subroutine for calculating monthly payments. Often, you may only want to calculate the monthly payment on a loan without calculating all of the rest of the parameters the entire program might furnish. In other words, you would like to be able to instruct your calculator to execute a single subroutine and terminate the calculation at the end of the subroutine.

The END key furnishes you with this capability. The END key is recognized by the calculator *only if the subroutine was called from the Idle mode with the EXECUTE SYMBOL key sequence*. If the subroutine is called during program execution, the calculator will ignore the END key.

When a subroutine is called from the Idle mode by the EXECUTE *SYMBOL* key sequence, the calculator terminates the calculation when it encounters the END instruction and returns to the Idle mode after resetting the program step counter to 0000.



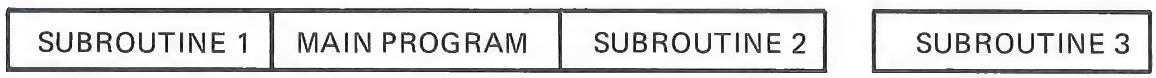
If the subroutine is called by the main program, the calculator ignores the END instruction and proceeds with the normal execution of the program.

If the RETURN ADDRESS, GO TO DISPLAY sequence is used to terminate the subroutine, the END keystroke should immediately precede these keystrokes. If the END keystroke doesn't precede these keystrokes, the calculator will execute the RETURN ADDRESS, GO TO DISPLAY sequence and never get to the END keystroke. Instead, it will branch to the address in the address register.

SUBROUTINES

SUBROUTINE LOCATION

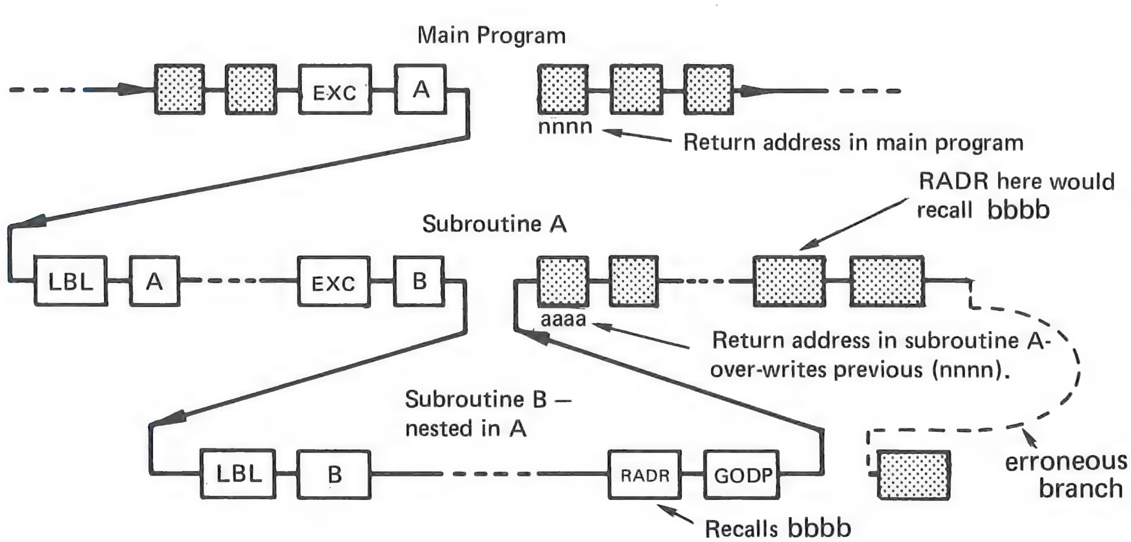
Subroutine placement is not restricted. Subroutines may be placed anywhere in memory; preceding, following, or in the main program.



If your program uses symbolically labeled subroutines that are to be called hundreds of times during the program execution, you may choose to place these subroutines near the beginning of the memory (location 0000). This will minimize the time the calculator uses to search for labeled subroutines. (Remember, when you call a symbolically labeled subroutine, the calculator begins its search at memory location 0000.)

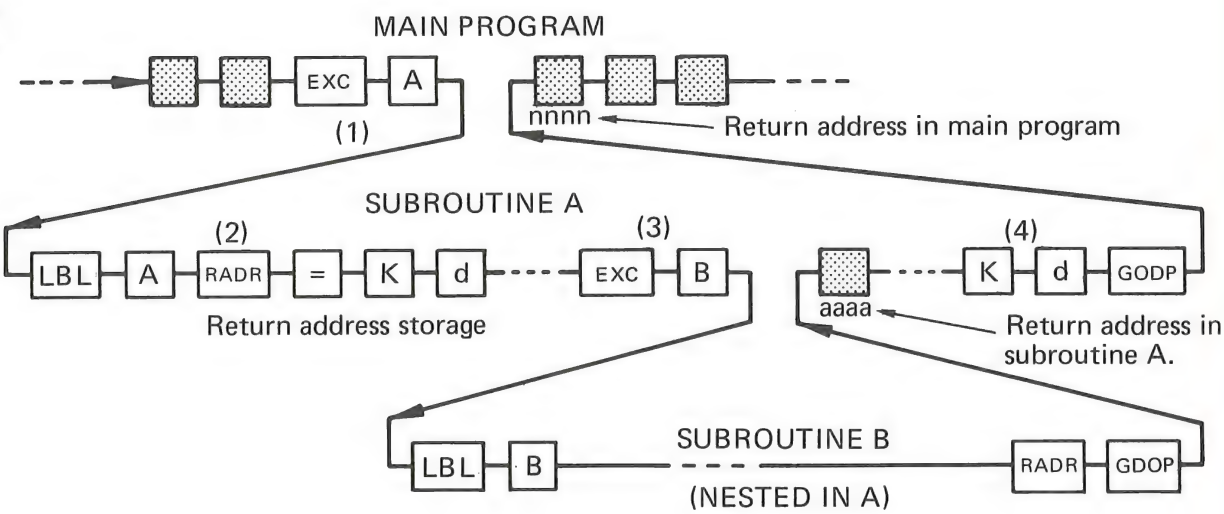
NESTING OF SUBROUTINES

Subroutines are said to be nested when program control is transferred from one subroutine to another before it is returned to the main program.



In the preceding illustration, when the calculator encounters the EXC A keystrokes in the main program, the return address for the main program is stored in the address register. In subroutine A, the EXC B keystrokes causes the return address for subroutine A to be stored in the address register, *OVER-WRITING THE RETURN ADDRESS FOR THE MAIN PROGRAM.*

When nesting subroutines, further provision must be made in order to return from the nested subroutines back to the main program. The simplest return address book-keeping method is to use the K or R registers to store the various return addresses. This technique is illustrated and explained below:



1) EXC A causes the return address in the main program to be stored in the address register.

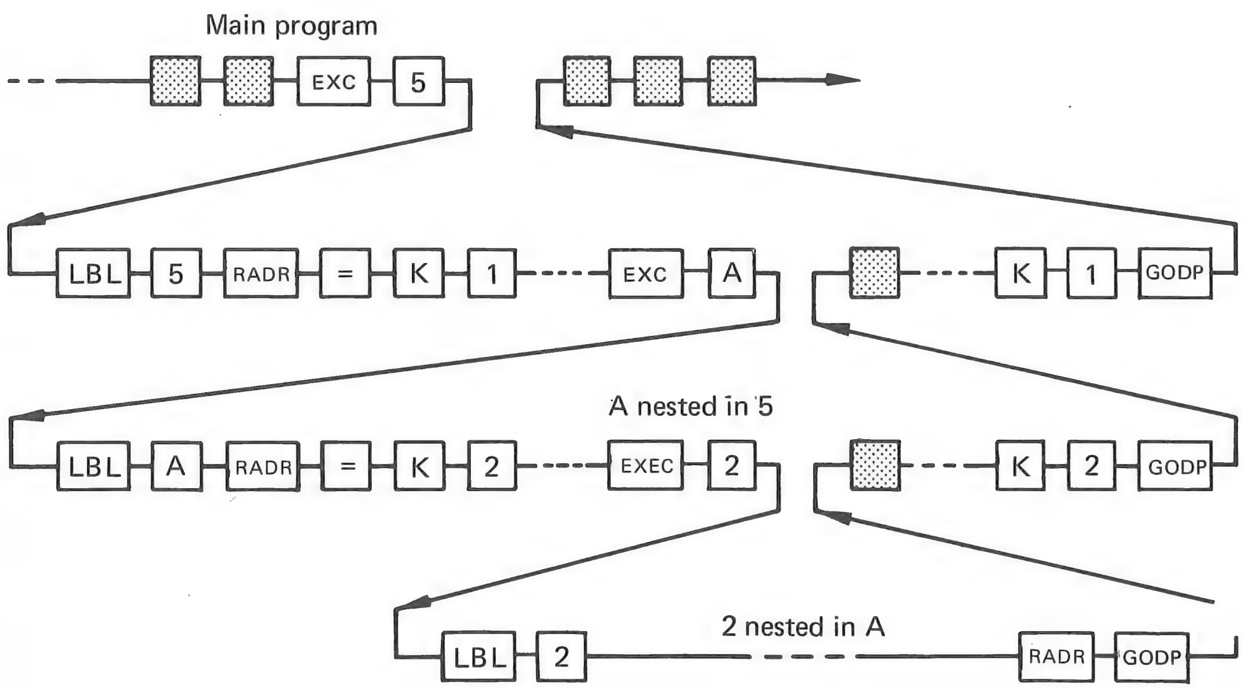
2) In subroutine A, the RADR causes the main program return address to be called into the display. This return address is then stored in K_d with the storage sequence, = K d.

SUBROUTINES

3) EXC B, in subroutine A, then causes the return address in subroutine A to be stored in the address register, and its current contents (main program return address) are lost.

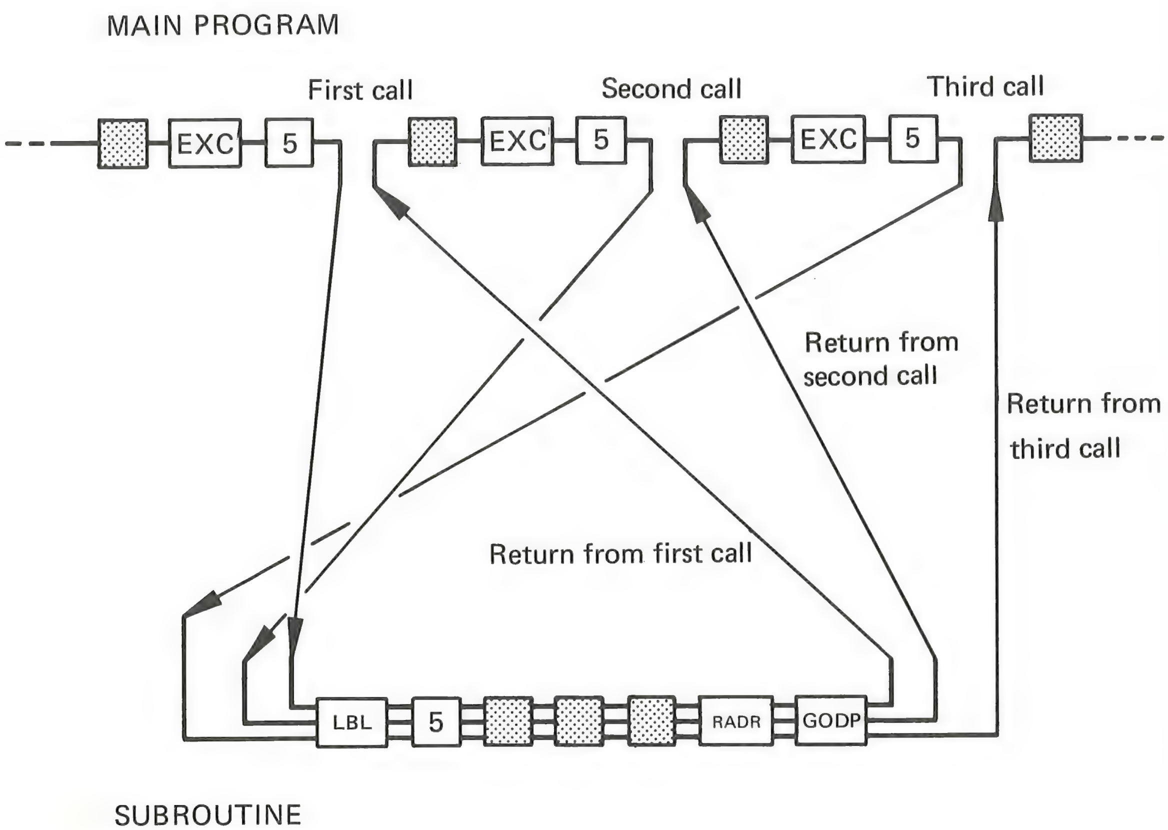
4) After the calculator completes subroutine A, the return address in the main program is recalled from K_d into the display and the GODP causes program control to be returned to the main program at the proper address.

The preceding example showed how to achieve single-level nesting: (only one subroutine was nested). However, there is no restriction, except memory size, on the number of subroutines that may be nested in a program. Multiple-level nesting may be achieved most simply by extending the method already given. This is illustrated below for two-level nesting; inspection will show that you may use this technique for more than two levels.



INTERACTIVE USE OF SUBROUTINES

The subroutines may be called any number of times in the main program. Each time the subroutine is called, a new address is placed in the return address register.



PROGRAMMING HINTS

A programmable calculator is most useful in performing the complex data manipulations that involve repetitious calculations, decision making, and branching. This is not to minimize the value of doing straight-forward calculations, but by now you are well aware of these capabilities. More exciting, however, are the potential applications of programming that couple straight-forward calculations with branching, looping, and decisions; these broaden your horizons in programming to include the kind of problems that, up until now, could only be solved on the more costly, large, and inaccessible computers. It follows, then, that if you wish to use the calculator as a computer with the above capabilities, you must understand how to write programs that accomplish these functions. The following discussion of programming hints is intended to give at least a basic understanding of the various means by which you may direct the calculator to accomplish true computer-like operations.

In order to demonstrate some of the basic concepts of programming as they relate to the calculator, let's teach the calculator how to count. We will start with a simple addition program and gradually expand it into a generalized counting program. Along the way we can demostrate some of the programming techniques that will help cement your comprehension of the whys and wherefores of programming.

As always though, before we actually start programming, we should examine the task at hand. Counting, you'll agree, is a simple process. Let's examine counting, *per se*, to see what the process might reveal. In this manner, as with all programs, we start at the beginning; a definition of the problem.

Before counting begins, we must first define a *starting point*, a *stopping point*, and a *counting increment*. The starting point tells us where to begin counting; the counting increment tells us what to count by (ones, twos, fives, etc.); and the stopping point is

where to stop counting. After we have this information, the counting *process* begins: Add the counting increment to the starting value and remember the sum. At this point, forget the starting value; all that is important now is the new value just obtained. Continue by adding the counting increment to the new value, always retaining the new value for the next addition. Each time a new sum is obtained, it must be compared to the stopping point. When the new value is equal to the stopping point, the count is complete.

Now let's teach the calculator how to count. Have the count start at zero and end at one hundred. Set the counting increment at one. Use the K_0 data register to store the numbers generated in the count. First, then, initialize K_0 to zero.

PRESS CLEAR DSPLY = K 0

Now, in a series of programmed instructions, take K_0, add one, and store the result in K_0. Display the result. Locate the program segment beginning at location 0100 in memory and make the segment a subroutine by beginning it with LABEL and the labeling keystroke, D/R.

PRESS

GO TO 0 1 0 0 LEARN	Enter the Learn Mode at 0100.
LABEL DEG RAD	The labeling keystrokes.
K 0 + 1 = K 0	Old value plus one stored in K_0.
STOP	Stop and display new value.
LEARN	Exit the Learn mode.

PROGRAMMING HINTS

Our program may be executed from the Idle mode by either of the four following methods:

1. PRESS [EXECUTE] [DEG RAD]

2. With an overlay installed, press the D/R key.

3. PRESS [LABEL] [DEG RAD] [CONT]

4. PRESS [GO TO] [0] [1] [0] [0] [CONT]

Any of the above keying sequences will cause execution of the program steps stored in memory at 0100, labeled with D/R. Each time this subroutine is executed, K_0, as evidenced by the display, will increment by one. Alternately, the program segment may be executed under program control by programming either of the two sequences given below.

PRESS	
[RESET] [LEARN]	Enter the Learn mode at 0000.
[EXECUTE] [DEG RAD] or [GO TO] [0] [1] [0] [0]	Either of these will work.
[LEARN]	Exit the Learn mode.

Notice that the first program requires two program steps, while the second requires five – herein lies one of the advantages of using subroutines.

Our instructions to the calculator now consist of a main program and a subroutine.

MAIN PROGRAM

```
0000   EXC_
0001   DG/R
```

SUBROUTINE

```
0100   LBL_
0101   DG/R
0102   K_
0103   0
0104   +
0105   1
0106   =
0107   K_
0108   0
0109   STOP
```

To execute the program press START. Notice that each time you do this, the number in the display, K_0, increments by one digit, as before. What happens is that the START keystroke causes execution of the program steps starting at 0000. The first program steps to be executed, then, are those that command execution of the subroutine D/R. The program then *branches* to 0100, the location of LABEL D/R, and executes these program steps, the last of which is STOP. Hence, the new value of K_0 is displayed and the *STOP* light comes on when the calculator finishes the subroutine. Another START will cause a repeat of the above process. But this is not very much good to us; we can count to 100 almost as fast as we can press START one hundred times. Let's program a START in the subroutine after the STOP.

PRESS

GO TO 0 1 1 0 LEARN	Enter the Learn mode at the location following the STOP.
START	The new program step.
LEARN	Exit the Learn mode.

PROGRAMMING HINTS

To run the program press START and subsequently press CONT. Remember that the STOP is still in the program and is executed as before. However, when CONT is pressed, START is executed, and as a result, the program *branches* from the location of the START to 0000, the beginning of memory. The subroutine is executed again, and the program again stops at the STOP. Press CONT to repeat. But this is not any better than before; now, because of the STOP, we just use CONT instead of START. Let's delete the STOP from the subroutine.

PRESS

GO TO 0 1 0 9 LEARN	Enter Learn mode at location of STOP.
DELETE	Delete step 0109, the STOP.
LEARN	Exit the Learn mode.

The main program and the subroutine now look like this.

MAIN PROGRAM

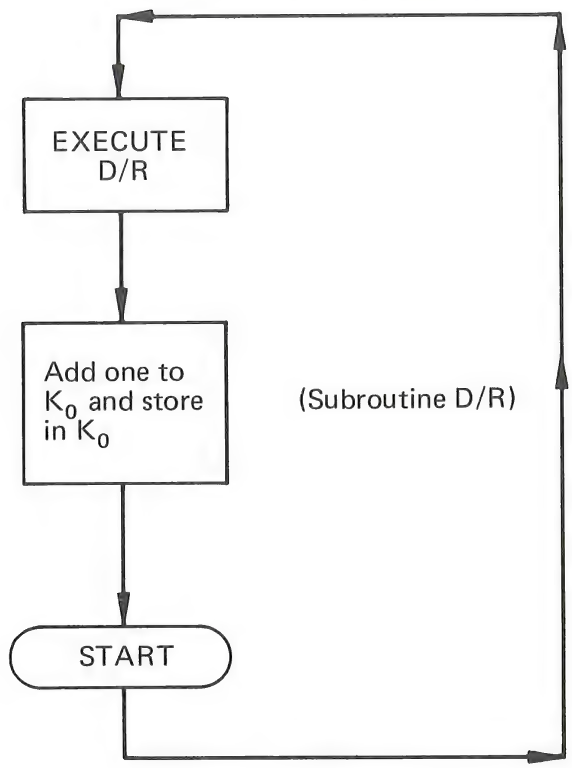
```
0000   EXC_
0001   DG/R
```

SUBROUTINE

```
0100   LBL_-
0101   DG/R
0102   K_
0103   0
0104   +
0105   1
0106   =
0107   K_
0108   0
0109   STRT
```

Notice that the START is now in the location that was previously occupied by the STOP; the DELETE reordered the program.

A flow chart of the program looks like this.



We have established a *loop* in the program; the START is an unconditional branch command causing program control to loop back to the beginning of the program and repeat the program steps in a continuing manner. What do you suppose will happen if we execute the program as it now stands? Try it and find out; press START.

As you might have expected, the calculator has entered the loop and is performing exactly as instructed by the program steps in the loop. The loop, as we've defined it, has a beginning but no end — we have thus far neglected to give the calculator any instructions about a stopping point. Therefore, unless we interrupt the loop with a manual STOP command, the calculator will continue counting ad-infinitum. By now the count has gone way past 100, our original goal. The program as it now stands can serve no more useful purpose than to demonstrate one of the pitfalls you might later encounter when programming. On a large computer a loop of this kind could be an expensive mistake — computer time is valuable.

PROGRAMMING HINTS

What we will do now is insert a *conditional branch* into the program. A conditional branch is similar to the unconditional branches (START and EXC D/R) we have been using except that the program will branch only when the proper *condition* is fulfilled. In our case that condition must be K_0 = 100, and we would like to display K_0 and STOP when the condition (K_0 = 100) is fulfilled. Otherwise, we would like to CONTINUE the count. The way we will achieve this is to compare K_0 to 100 and when K_0 is equal to 100, we will stop.

Program the following keystrokes starting at location 0109, the present location of the programmed START command.

PRESS

GO TO ▪▪▪▪ 0 1 0 9 LEARN	Enter the Learn mode at the location of the START.
− 1 0 0)	Subtract 100 from K_0.
=ø	Is $K_0 - 100 = 0$?
K 0 STOP	If K_0 = 100, display K_0 and stop.
CONT	If K_0 is *not* equal to 100, continue.
START	Loop to 0000 and resume the count.
LEARN	Exit the Learn mode

Start the program by again initializing K_0 to zero and press START. As a result, the loop will be executed until K_0 is equal to 100, at which time the calculator will recognize this fact (in the program steps we have just entered) and STOP with the desired value of K_0 in the display.

PROGRAMMING HINTS

The complete program now looks like this.

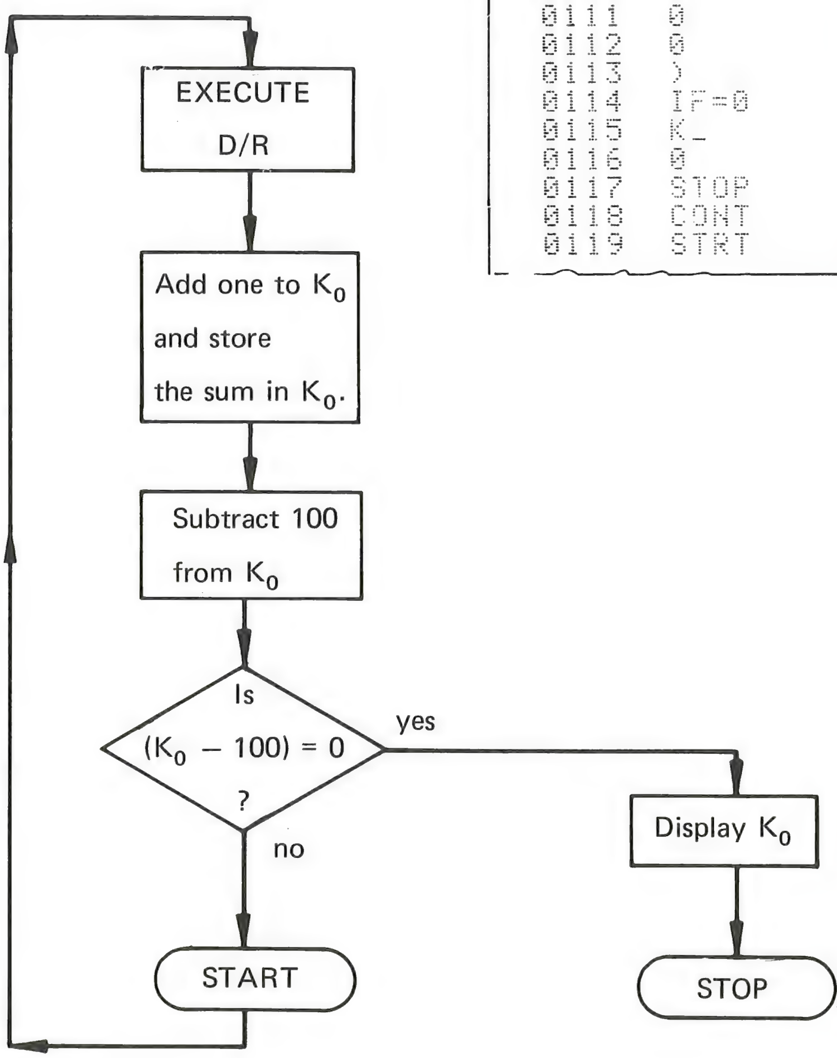
MAIN PROGRAM

```
0000   EXC_
0001   DG/R
```

SUBROUTINE

```
0100   LBL_
0101   DG/R
0102   K_
0103   0
0104   +
0105   1
0106   =
0107   K_
0108   0
0109   -
0110   1
0111   0
0112   0
0113   )
0114   IF=0
0115   K_
0116   0
0117   STOP
0118   CONT
0119   STRT
```

The flow chart looks like this.



PROGRAMMING HINTS

In the flow chart notice how the IF statement is diagramed. The diamond shape is always used to indicate a question when flow charting a program. There are two exit paths from the diamond; the *yes* path (yes, the condition is fulfilled), and the *no* path (the condition is *not* fulfilled). Note that there are five IF CONDITIONS on the calculator; each of them may be used whenever required.

But still, the calculator does not know enough — it can only count by ones and it can only count to one hundred. This is not very impressive. In addition, we have to re-initialize the starting point each time we begin the count — not very convenient. Let's alter the program so that the starting point, the counting increment, and the stopping point are all entered into the program *as part of* the program. In this manner, the program becomes *interactive* with you, the programmer.

First, then, rewrite the main program so that it calls for execution of an *initialization* subroutine that we will call 'arc'.

PRESS	RESET LEARN	Enter the Learn mode at 0000.
	EXECUTE arc	Call for execution of subroutine arc.
	EXECUTE DEG RAD	Call for execution of subroutine D/R.
	LEARN	Exit the Learn mode.

PROGRAMMING HINTS

We must now write the initialization subroutine. But before we do this, we must alter the counting subroutine, D/R. Enter it according to the keystrokes below.

NEW SUBROUTINE

OLD SUBROUTINE

LBL D/R

K 0 + 1 —— (K$_1$ is the new counting increment.) ——→

= K 0

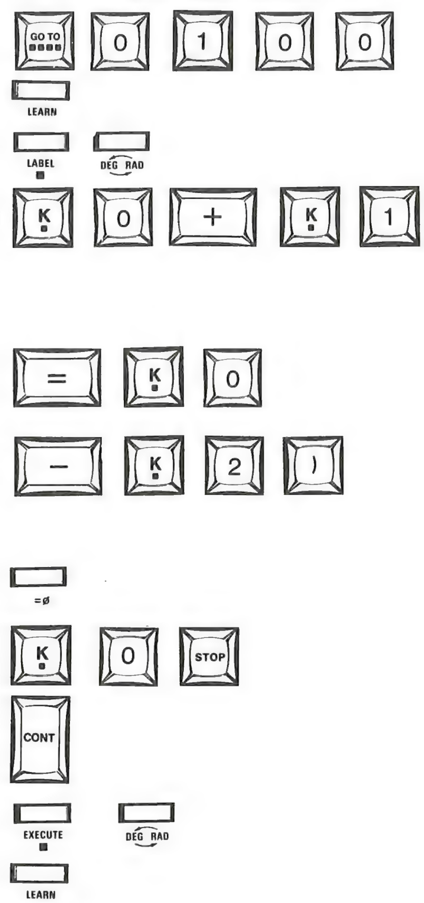
– 1 0 0) —— (K$_2$ is the new stopping point.) ——→

IF=0

K 0 STOP

CONT

START —— (The new looping command.) ——→



In the above we have replaced the old counting increment, one, with a *variable* counting increment, K_1. Also, the old stopping point, one hundred, is replaced with a *variable* stopping point, K_2. The old starting value, K_0, remains the same, but note that the START in the old subroutine is replaced by EXECUTE D/R; the subroutine is now calling for execution of itself. (This is perfectly valid, but remember what will happen to the return address.)

PROGRAMMING HINTS

Now we can write the initialization subroutine in which the starting point (K_0), the counting increment (K_1), and the stopping point (K_2), will each be initialized and stored for later recall. Put the new subroutine in memory starting at location 0200.

PRESS

GO TO ■■■■ 0 2 0 0 LEARN LABEL ■ arc	Enter the Learn mode at 0200.
CLEAR DSPLY STOP	Clear the display and stop.
= K ■ 0	Store display, starting point, in K_0.
CLEAR DSPLY STOP	Clear the display and stop.
= K ■ 1	Store display, counting increment, in K_1.
CLEAR DSPLY STOP	Clear the display and stop.
= K ■ 2	Store display, stopping point, in K_2.
RETURN ADDRESS	Put the return address into the display.
GO TO DISPLAY	Direct program control to the location shown in the display.
LEARN	Exit the Learn mode.

PROGRAMMING HINTS

To execute the program press START. The program will STOP with zeros in the display. Enter the starting value, the counting increment, and the stopping point. After each entry press CONTINUE. When CONTINUE is pressed the third time, the program will run as before.

SAMPLE EXECUTION

PRESS START

Enter 0	CONT	Starting point
1	CONT	Counting increment
1 0 0	CONT	Stopping point

The program will run and stop with 100 in the display.

Now try to run the program with a counting increment of three, and everything else as before. What happens? The calculator is again in a *loop*. The reason, of course, is that we have overlooked one detail — namely, it is not possible to count to exactly 100 by threes with our program! Starting at zero, one hundred is not an integer-multiple-sum of three. Hence, the branching condition, ($K_0 - K_2 = 0$) is never met. Therefore we must change the program so that it exits the loop when the value of the count is *greater than or equal to* the stopping point. To do this, replace the present IF condition (IF=0) with IF$\geqslant$0.

PRESS

GO TO 0 1 1 4 LEARN	Enter the Learn mode at the location of the IF statement.
≥ø	The new IF condition
LEARN	Exit the Learn mode

PROGRAMMING HINTS

The complete program now looks like this:

MAIN PROGRAM

```
0000   EXC_
0001   ARC
0002   EXC_
0003   DG/R
```

INITIALIZATION SUBROUTINE

```
0200   LBL_
0201   ARC
0202   CLDP
0203   STOP
0204   =
0205   K_
0206   0
0207   CLDP
0208   STOP
0209   =
0210   K_
0211   1
0212   CLDP
0213   STOP
0214   =
0215   K_
0216   2
0217   R AD
0218   GODP
```

COUNTING SUBROUTINE

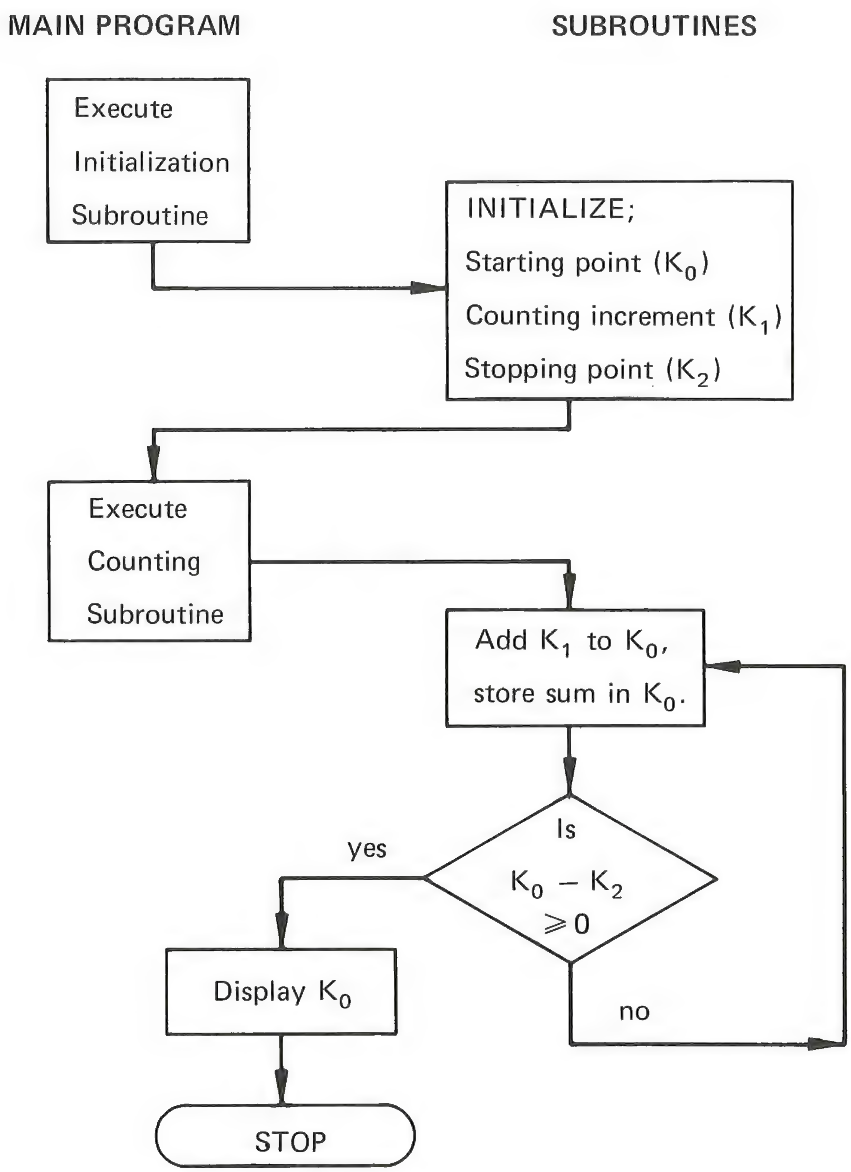
```
0100   LBL_
0101   DG/R
0102   K_
0103   0
0104   +
0105   K_
0106   1
0107   =
0108   K_
0109   0
0110   -
0111   K_
0112   2
0113   )
0114   IF>=
0115   K_
0116   0
0117   STOP
0118   CONT
0119   EXC_
0120   DG/R
```

Now try the program with a counting increment of three:

PRESS	START	
	0 CONT	Enter the starting point.
	3 CONT	Enter the counting increment.
	1 0 0 CONT	Enter the stopping point.

The program will stop with 102 in the display; 102 is the least multiple integer sum of three that is greater than or equal to 100.

A flow chart of our counting program looks like this:



PROGRAMMING HINTS

The calculator now knows how to count. However, it does make some rather simple assumptions. For instance, what will happen if you tell it to count from zero to a negative number with a positive counting increment? What will happen if you give it a negative counting increment with positive starting and stopping points? As you might guess, the calculator takes your inputs quite literally, and performs *only* the program steps it has in its memory. In the first instance, it only counts once — up to the counting increment. In the second instance, we *again* enter an unterminated loop as the calculator, with a negative counting increment, will count *backwards* from the starting point and never reach the positive stopping point.

Thus we see that the calculator, quite literally, must be taught everything that is does not already know. In addition to this, since programmable memory is *volatile*, the calculator will *forget* all that you have taught it whenever AC power to it is even momentarily interrupted. This suggests that you record, on mag-tape, any program that you feel is worth repeating at some later date. Let's go through this exercise and record the program we have just written.

1) Insert a mag-tape cartridge into the tape transport mechanism. (Don't forget to first insert the write-enable button.)

2) Record the program on block zero of the six possible blocks.

   PRESS [RESET] [TO TAPE] [0]

   You now have a permanent record.

Now, when you come back later, you can re-enter the counting program by following the simple sequence,

[RESET] [FROM TAPE] [0]

If you wish to put the program into memory at a non-0000 location, simply replace the RESET with the sequence,

ADDRS GO TO d d d d

Heretofore, we have discussed three of the most important building blocks of computer programming: looping, branching, and decision making. Each of these were used in the preceding discussions, and each performed a unique purpose. If you have read all the preceding sections of this manual and understand them, you now possess all the tools required to become a successful programmer, provided that you recognize the fact that it is not only the tools, but also techniques that are required. In this sense you have access to the tools (all the functions on the calculator) but the techniques are of two disciplines: programming and mathematics. The mathematics involved in computer programming range from the simple to the elegant. Here we will keep on the simple side, but this does not rule out application of more elegant use of the language. Now for programming: to iterate the idea given in the introduction, programming is nothing more than organized thought. What is required here is not so much work as you might at first believe. Rather, the most important ingredient is a willingness to sit down and really think about the problem at hand. Once you think you understand how to solve the problem, by whatever methods you have available, then is the time to think about programming the solution. Remember, programming is organization, and a program is a series of steps, each chipping away at a problem. The chips that fall do not of themselves mean anything—but it is the order and *which* chips that fall that finally reduces a granite problem to a sculptured solution.

PROGRAMMING HINTS

LOAN CALCULATION PROGRAM

Oftentimes we are confronted with financial problems that unless wisely resolved will result in undue strain to our usually tenuous budgets. Most financial problems involve nothing more than juggling, but at times, like when we want to buy a house or car, the information we need is simply not available unless we consult the local bank *or* have a smart calculator at hand to answer our financial questions. Typical loan questions are:

- What is the monthly payment on a new loan?
- How much was the principal (amount owed) reduced on a loan during the last N periods?
- How much of the last payment was applied to the principal?
- How much of the last payment was interest?

Let's write a program that will answer these questions. First, the problem must be defined. The equation that gives the payment per period on an amortized loan is given below.

$$A = \frac{P\,i}{1 - (1 + i)^{-n}}$$

(A is rounded up.)

A = Payment amount, dollars per period.

P = Principal amount, dollars

i = Interest rate per period, expressed as a decimal

n = Number of payment periods

A period is the time between loan payments and may be days, months, or years.

PROGRAMMING HINTS

When loan payments are computed, they consist of a principal payment and an interest payment. The interest payment is computed as the interest rate *times* the remaining principal. The interest payment is rounded *up*.

The interest per period is expressed as

Interest payment per period, $A_i = i\ P$ (A_i is rounded up) (Eq 1)

The principal payment per period is then computed by subtracting the interest payment (A_i) from the payment per period (A). This is expressed as

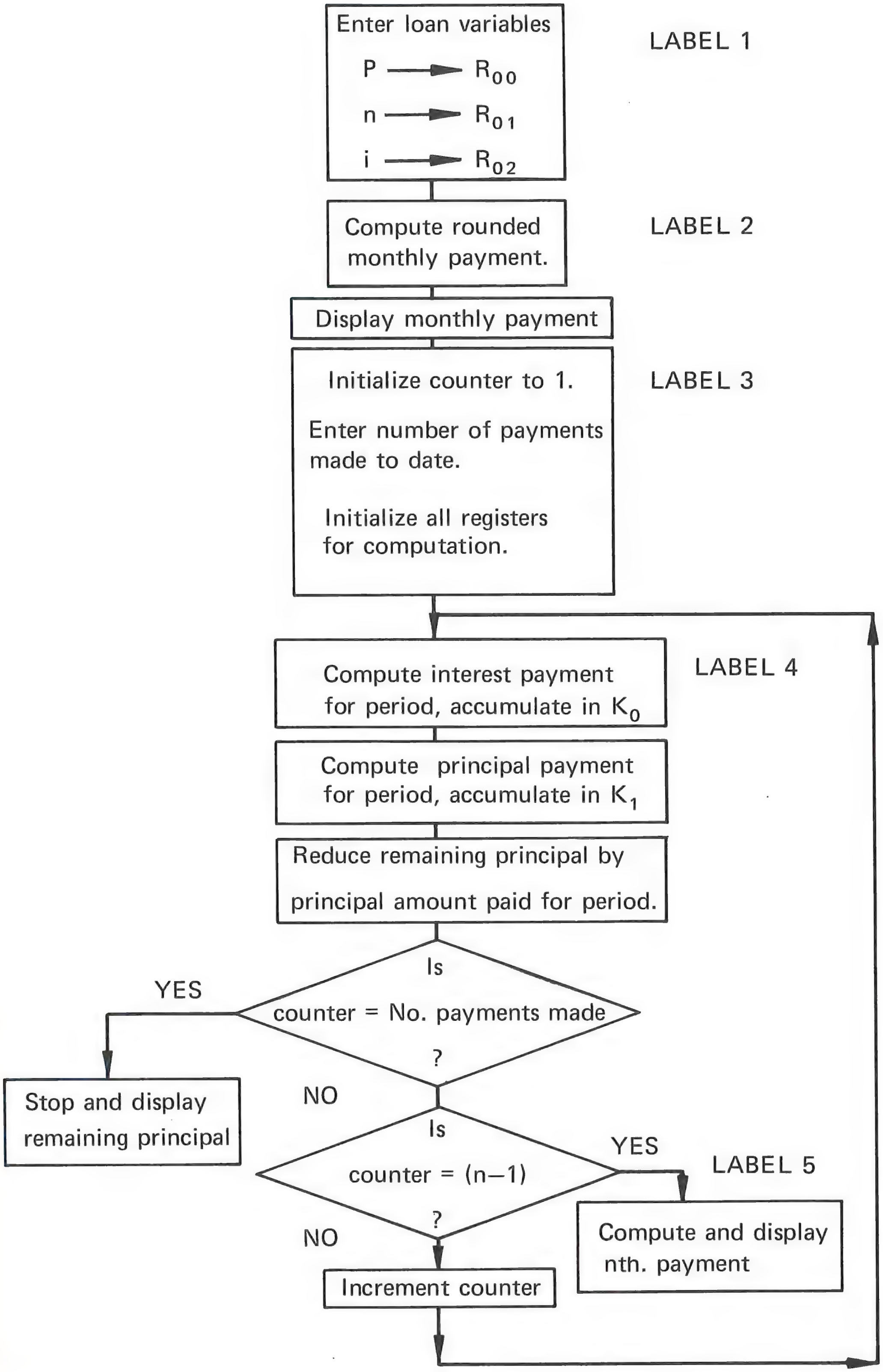
Principal payment per period, $A_p = A - A_i$ (Eq 2)

When a payment, A, is made for a given period, the principal is reduced by A_p, the principal payment for the period. When the next payment is made, the interest payment, A_i, is then computed using the reduced principal. In this manner, as the loan progresses, the total payment remains constant. However, the amount paid in interest and principal changes as the remaining principal is reduced during each period. The last payment is not A, but is computed so that the remaining principal is reduced to zero after the interest is computed.

Now let's program the calculator to do computations on the amortized loan. Compute the payment per period (A), given the principal amount (P), the interest rate (i), and the number of payment periods (n). For simplicity, let's enter the *yearly* interest rate and use monthly payments.

Study the following flow chart to determine how this program is organized — subroutines are indicated on the flow chart. Enter the main program into memory, starting at 0000. The subroutines may be placed anywhere in the memory.

PROGRAMMING HINTS



PROGRAMMING HINTS

MAIN PROGRAM	Start the main program at 0000
[EXECUTE] [1]	Call subroutine for loan variable entry
[EXECUTE] [2]	Call subroutine to compute monthly payment
[R] [0] [3] [STOP]	Stop and display monthly payment
[EXECUTE] [3]	Call subroutine for amortization
[LABEL] [1]	Subroutine for loan variable entry
[CLEAR DSPLY] [STOP] [=] [R] [0] [0]	Enter principal, store in R_{00}
[CLEAR DSPLY] [STOP] [=] [R] [0] [1]	Enter term of loan in months, store in R_{01}
[CLEAR DSPLY] [STOP] [÷] [1] [2] [0] [0] [=] [R] [0] [2]	Enter yearly interest rate in percent, convert to decimal monthly rate, store in R_{02}
[RETURN ADDRESS] [GO TO DISPLAY]	Return to main program

PROGRAMMING HINTS

LABEL 2	Subroutine to compute monthly payment
RETURN ADDRESS = K 9	Store return address in K_9
CLEAR DSPLY 1 + R 0 2)	(1 + i)
$\lvert x \rvert^a$ R 0 1 +/-)	$(1 + i)^{-n}$
+/- + 1) $\frac{1}{x}$	$(1 - (1 + i)^{-n}\,{}^{-1}$
× R 0 0 × R 0 2)	Exact monthly payment
EXECUTE int x	Call for rounding subroutine
R 5 0 = R 0 3	Store rounded payment in R_{03}
K 9	Recall return address from K_9
GO TO DISPLAY	Return to main program
LABEL 3	Initialization subroutine
R 0 0 = R 1 0	Establish remaining principal
CLEAR DSPLY = K 0 = K 1	Initialize K_0 and K_1 to zero
1 = K 2 = K 3	Initialize K_2 and K_3 to one
CLEAR DSPLY STOP = K 4	Enter number of payments made to date
EXECUTE 4	Transfer control to subroutine 4

PROGRAMMING HINTS

LABEL 4	Subroutine to compute amortization
R 1 0 × R 0 2)	Monthly interest, exact
EXECUTE $int\,x$	Call for rounding subroutine
R 5 0 Σ_0	Sum interest in K_0
+− + R 0 3) Σ_1	Principal payment, summed in K_1
+− + R 1 0 = R 1 0	Compute remaining principal
K 2 − K 4)	Compare counter to number of payments made
=0 R 1 0 STOP EXECUTE 3 CONT	If counter is equal to number of payments made, display remaining balance. A manual CONT will cause a branch to LBL 3.
K 2 − (R 0 1 − 1))	Compare counter to (n — 1)
=0 EXECUTE 5 CONT	If counter = (n — 1), next payment is last payment; execute subroutine 5 to compute last payment.
$_3\Sigma_2$	Increment counter
EXECUTE 4	Loop to beginning of subroutine for another calculation.

PROGRAMMING HINTS

LABEL 5	Subroutine to compute last payment
R 1 0 × R 0 2)	Last interest payment, exact
EXECUTE $int\ x$	Call for rounding subroutine
R 5 0 Σ_0	Accumulate in K_0
+ R 1 0 Σ_1	Last Principal accumulated in K_1
) STOP	Last payment displayed
LABEL $int\ x$	Rounding subroutine
× 1 0 0	Multiply by 100
+ 3 $\frac{1}{x}$ × 3)	Add .999999999999
$int\ x$ ÷ 1 0 0	Take integer value and divide by 100
= R 5 0	Store in R_{50}
RETURN ADDRESS GO TO DISPLAY	Return to calling point

PROGRAMMING HINTS

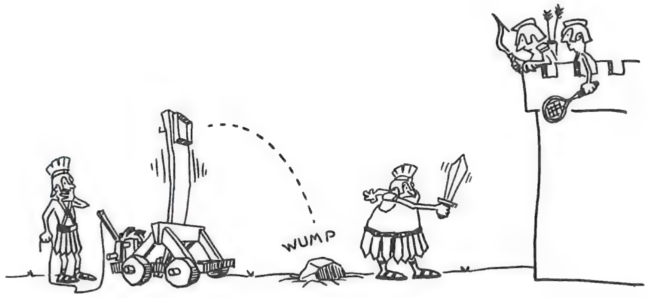
SAMPLE EXECUTION

Say you want to buy a house that costs $20,000 and the present annual interest rate on a 25 year, 90% mortgage is 7.5%. What are the monthly payments on this loan? After one year, what is the remaining balance on the loan? How much interest will be paid during this period?

1. PRESS [START]
2. Enter [1] [8] [0] [0] [0] [CONT] — Amount borrowed
3. Enter [3] [0] [0] [CONT] — Term of loan in *months*
4. Enter [7] [.] [5] [CONT] — Yearly interest rate, %
5. The monthly payment is $133.02
6. PRESS [CONT]
7. Enter [1] [2] [CONT] — Number of payments made on loan
8. The remaining balance after twelve payments is $17745.19
9. The total interest paid during this period is in register K_0: $1341.43
10. The total principal paid during this period is in register K_1: $254.81

If you desire to compute the interest expense *between* two periods, enter the first period at step 7 and record the interest paid when this computation is complete. Then press CONT — since the next programmed instruction after the STOP in subroutine LABEL 4 is EXECUTE 3, the program will loop back to the beginning of the initialization subroutine, LABEL 3, and here you can enter a new period. When the computation is again completed, simply find the difference between the interest expenses.

PROGRAMMING HINTS



A BALLISTICS PROBLEM

The word ballistics, with roots in the Greek verb *ballin* – to throw, today means the study of moving projectiles and is derived from the ancient Roman use of the term *ballista* to label what we now know as the catapult. Before their obsolescence by cannon in the Middle ages, catapults were developed with power sufficient to hurtle projectiles weighting 300 pounds a distance exceeding 300 yards. This translates to a launch velocity of about 170 feet per second.

As an exercise in programming let us construct and solve a hypothetical ballistics problem that you might have encountered in the role of a Middle Ages artillery sergeant participating in the seige of a walled fortress held by the nasty and cruel Feudal Lord. We make two assumptions: 1) a Newtonian understanding of trajectories, and 2) possession of a Tektronix calculator.

THE PROBLEM: The enemy defenders are on top of a wall, h feet high and t feet wide, located a distance, d, from your catapult. Your army is in front of the wall and attempting to scale it with ladders. Your task is to hit the top of the wall by determining a launch angle in degrees. You know the size of the wall and its location, along with the initial velocity of your projectile in feet per second.

PROGRAMMING HINTS

The trajectory, y, is a function of:

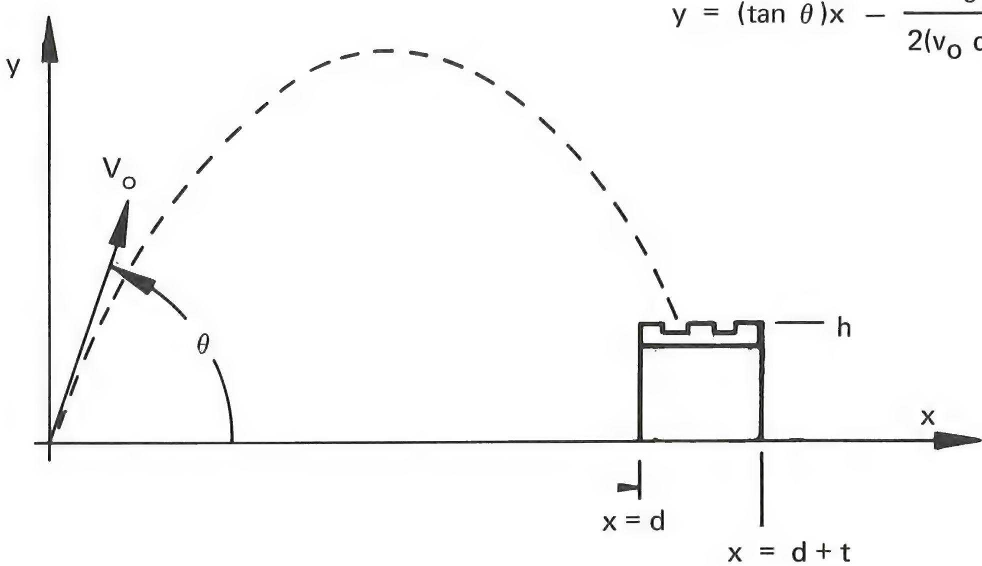
1) Launch angle, θ

2) Initial velocity, V_o

3) Force of gravity, $g = 32\ \text{ft/sec}^2$

4) Distance, x, from the launch point

$$y = (\tan\theta)x - \frac{g\,x^2}{2(v_o\cos\theta)^2}$$



The trajectory equation is quadratic and of the form,

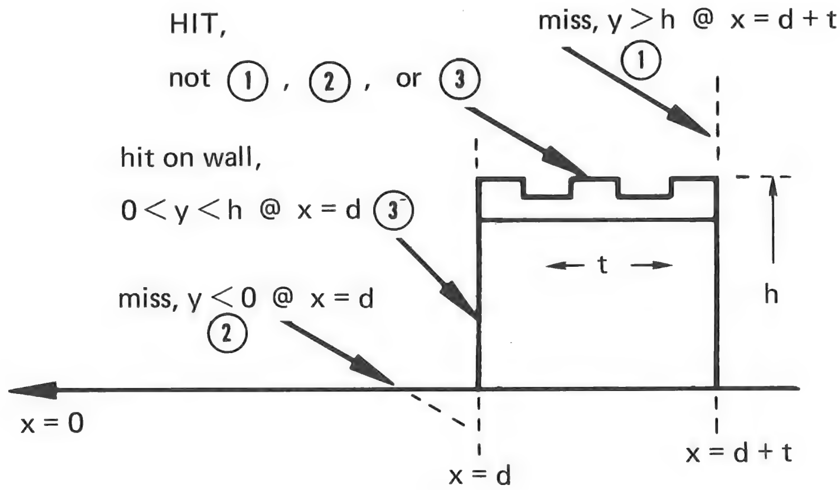
$$A\,x^2 + B\,x + C = 0$$

At y = 0, C = 0, and the point of impact is

$$x_{impact} = \frac{B}{A}$$

PROGRAMMING HINTS

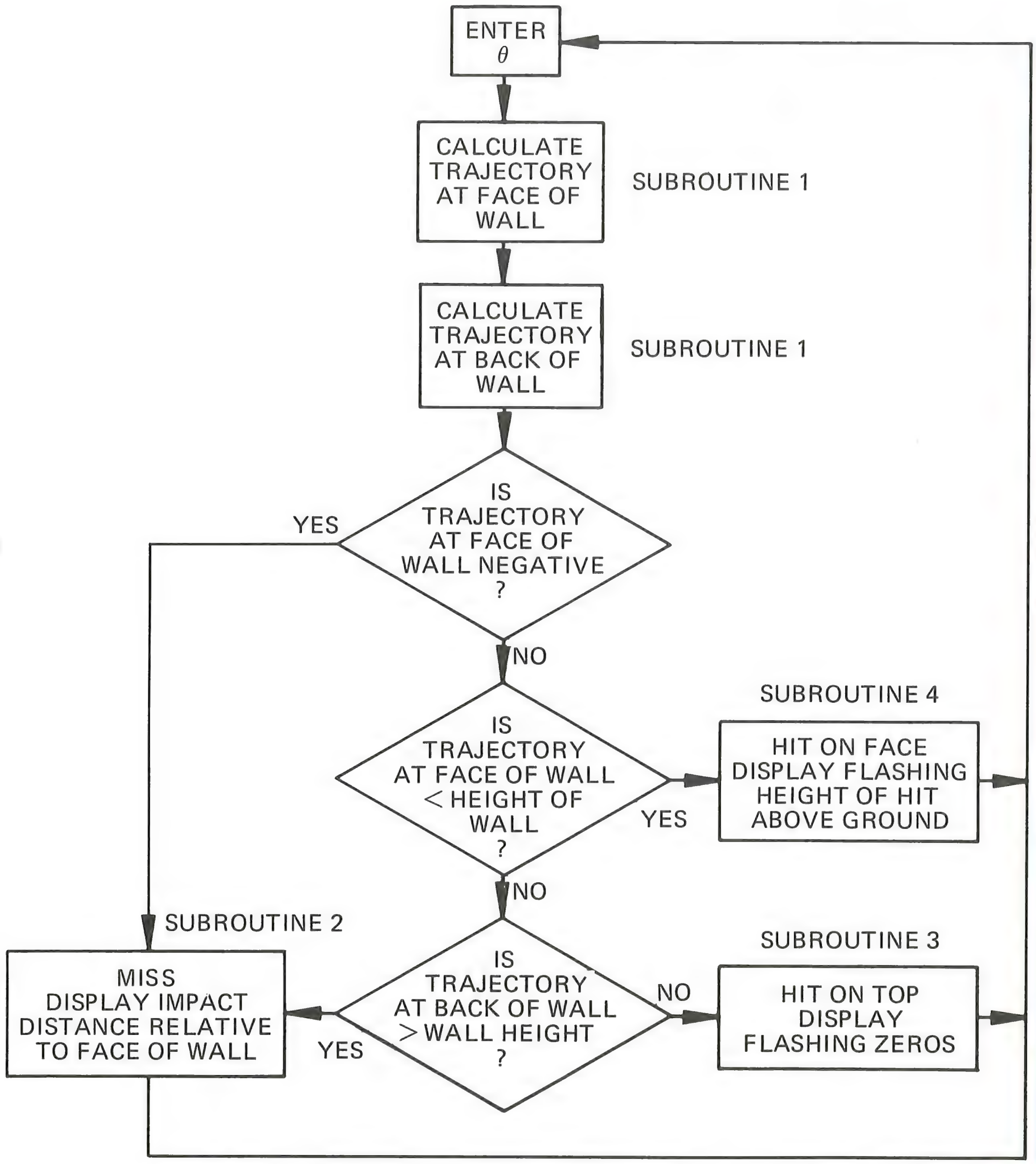
Study the figure below to determine hit-miss conditions.



To solve the problem, first manually enter all variables except the launch angle:

Initial velocity, v_0, in feet per second	= K 0
Distance to wall, d, in feet	= K 1
Height of wall, h, in feet	= K 2
Thickness of wall, t, in feet	= K 3

Start the programming by entering the launch angle, Θ; store in K_4. Evaluate the trajectory at x = d and x = (d + t). If the projectile hits the top of the wall, display flashing zeros. If it hits the face of the wall, display (flashing) the height of the hit above ground. If the projectile misses the wall, calculate and display the impact distance relative to the face of the wall; negative display for impact in front of the wall, and positive display for an impact behind the wall. Provide for looping so that new angles may be entered easily for subsequent calculations.


ENTER
θ
CALCULATE TRAJECTORY AT FACE OF WALL
SUBROUTINE 1
CALCULATE TRAJECTORY AT BACK OF WALL
SUBROUTINE 1
IS TRAJECTORY AT FACE OF WALL NEGATIVE ?
YES
NO
SUBROUTINE 4
IS TRAJECTORY AT FACE OF WALL < HEIGHT OF WALL ?
YES
HIT ON FACE DISPLAY FLASHING HEIGHT OF HIT ABOVE GROUND
NO
SUBROUTINE 2
SUBROUTINE 3
MISS DISPLAY IMPACT DISTANCE RELATIVE TO FACE OF WALL
IS TRAJECTORY AT BACK OF WALL > WALL HEIGHT ?
YES
NO
HIT ON TOP DISPLAY FLASHING ZEROS

PROGRAMMING HINTS

MAIN PROGRAM

Keys	Description
CLEAR DSPLY · STOP · = · K · 4	Store displayed launch angle in K_4
K · 1 · = · R · 0 · 0 · 3	Set x at d
EXECUTE · 1	Call for trajectory subroutine
R · 2 · 2 · = · R · 0 · 1	y(d) stored in R_{01}
K · 1 · + · K · 3 · = · R	Set x at (d + t)
0 · 3	
EXECUTE · 1	Call for trajectory subroutine
R · 2 · 2 · = · R · 0 · 2	y(d + t) stored in R_{02}
R · 0 · 1	Recall y(d)
<0 · EXECUTE · 2	If y(d) negative, calculate miss
CONT	Otherwise, continue
R · 0 · 1 · − · K · 2 ·)	Compare y(d + t) to h
<0 · EXECUTE · 4	If y(d + t) $\geqslant$ h, calculate miss
CONT	Otherwise, continue

MAIN PROGRAM (cont)–

R 0 2 − K 2)	Compare y(d) to h
≥∅ EXECUTE 2	If y(d) ⩾ h, hit on top
CONT	Otherwise, continue
EXECUTE 3	Hit on face

SUBROUTINES

LABEL 1	Trajectory subroutine
K 4 $\cos x$ × K 0) x^2	$(v_o \cos \theta)^2$
× 2) $\frac{1}{x}$	$\left[2(v_o \cos \theta)^2\right]^{-1}$
× R 0 3 x^2 × 3 2	Complete x^2 term
+⁄−)	
+ K 4 $\tan x$ × R 0 3)	y(x), the trajectory
= R 2 2	Store in R_{22}
RETURN ADDRESS GO TO DISPLAY	Return to main program

PROGRAMMING HINTS

LABEL 2 CLEAR DEG RAD	Impact subroutine, for miss
K 4 cos x × K 0)	$(v_0 \cos \theta)^2$
x^2	
× K 4 tan x × 2)	$2 \tan \theta \ (v_0 \cos \theta)^2$
÷ 3 2)	Impact relative to x = 0
− K 1) STOP	Display impact relative to face of wall
START	Loop to beginning of program
LABEL 3	Subroutine for hit on top of wall
CLEAR DSPLY $\frac{1}{x}$ CLEAR DSPLY STOP	Display flashing zeros
CLEAR	Reset error message
DEG RAD	Re-establish degree operation
START	Loop to beginning of program
LABEL 4	Subroutine for hit on face of wall
CLEAR DSPLY $\frac{1}{x}$ R 0 1 STOP	Display height of hit above ground (flash)
CLEAR DEG RAD START	Reset error message, establish degree operation, and loop to beginning of program

SAMPLE EXECUTION

Hit the top of a wall that is 20 feet high and 10 feet thick. The wall is located 120 feet away from the catapult which has a launch velocity of 100 feet per second.

First, enter the constants, v_O, d, h, and t, into their assigned registers.

CLEAR DSPLY 1 0 0 = K 0

CLEAR DSPLY 1 2 0 = K 1

CLEAR DSPLY 2 0 = K 2

CLEAR DSPLY 1 0 = K 3

Now press CLEAR and D/R so as to establish degree operation.

PRESS	ENTER		DISPLAY	
START	10 CONT		−13.11	(hit in front)
CONT	15 CONT	Flashing	7.45	(hit on face)
CONT	20 CONT	Flashing	17.58	(hit on face)
CONT	21 CONT	Flashing	19.62	(hit on face)
CONT	22 CONT		97.08	(hit background)
CONT	21.5 CONT	Flashing	zeros	(hit on top)
CONT	80 CONT		−13.11	(hit on front)
CONT	79 CONT		− 2.93	(hit in front)
CONT	78 CONT	Flashing	zeros	(hit on top)

Thus, the two angles that will hit the top of the wall are 21.5 and 78 degrees.

PROGRAMMING

PROGRAMMING HINTS

RANDOM NUMBERS AND GAMES

Most computer games, whether they are based on chance or probability, must by necessity include a random number generator. The one we will use here is based on the equation given below.

$$x_n = (x_{n-1} + \pi)^8 - \text{integer } (x_{n-1} + \pi)^8 \quad 0 \leqslant x_n < 1$$

To generate a random number between zero and nine, simply take the value of x_n, multiply it by 10, and take the integer value of the product.

Here is a program that will generate random numbers between 0 and 9.

"The following subroutines will be used in later examples"

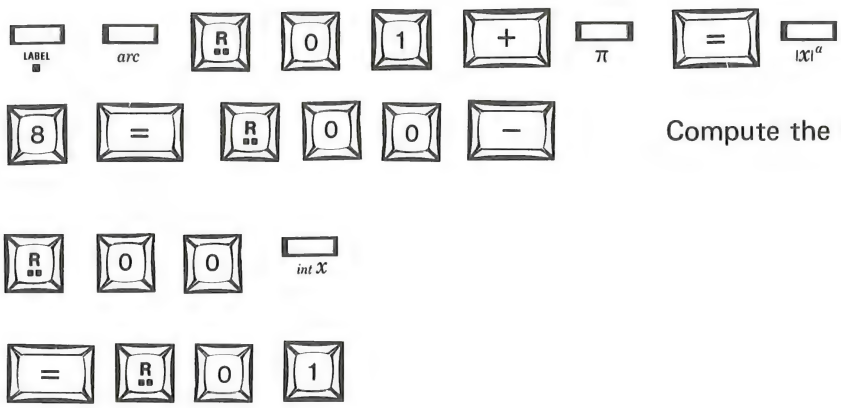
0200 MAIN PROGRAM

Keys	
EXECUTE DEG RAD	Call for execution of subroutine that contains random number generator. Display random number and stop.
R 0 1 RESET	

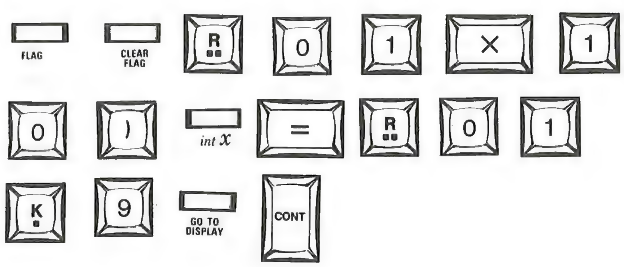
0100 SUBROUTINES

Keys	
LABEL DEG RAD	Beginning of subroutine
RETURN ADDRESS = K 9	Store return address in K_9
CLEAR DSPLY = R 0 0 0 =	
R 0 1	Initialize constants

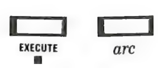
PROGRAMMING HINTS



Compute the random number



Check to see if the FLAG is set, and if it has been set, recall the return address into the display and exit the subroutine.



Loop to LBL arc

To execute the program press START. After a few seconds press STOP, SET FLAG, and CONT. The result that appears in the display is a number from zero to nine that is randomly dependent on the time that you waited to press STOP. To view all the random numbers as they are computed, insert a STOP or a PRINT DISPLAY before the IF statement.

PROGRAMMING HINTS

SOME SIMPLE CALCULATOR GAMES

E S P

One simple game you might wish to program is called ESP. Program the calculator to accept a number of your choice (0 through 9). Store the chosen number in a data register and then call the random number generator subroutine. Upon return to the main program compare the chosen number with the random one; if they match, indicate with a flashing display. If no match occurs display the random number and guess again.

0000 MAIN PROGRAM

[CLEAR DSPLY] [STOP] [=] [K] [1]	Stop to enter choice of number
[EXECUTE] [DEG RAD]	Execute random number subroutine
[K] [1] [−] [R] [0] [1] [)]	Compare choice with random number
[=Ø] [CLEAR DSPLY] [$\frac{1}{x}$] [CLEAR DSPLY] [CONT]	If they match, flash display
[R] [0] [1] [RESET]	If no display random number

To run the program press START. The STOP Mode will immediately be indicated. Enter your choice of digits 0 to 9. Wait for a few seconds and press STOP, SET FLAG, and CONT. If you have guessed right your choice will appear flashing in the display. If you guessed wrong, the random number will appear in the display.

ASK SWAMI

If you have a printer you can program the calculator to answer yes-no questions (randomly). Generate a random number and upon return to the main program, execute, according to the random number, any of ten subroutines that contain various forms of yes and no answers as listed below.

Label	Printout (yes)	Label	Printout (no)
0	TO BE SURE	5	NOT AT ALL
1	MOST ASSUREDLY	6	NOT IN THE LEAST
2	CERTAINLY	7	IMPOSSIBLE
3	DOUBTLESS	8	BY NO MEANS
4	UNQUESTIONABLY	9	NAY

MAIN PROGRAM

0000 EXECUTE DEG RAD R 0 1 × 2 + 4) GO TO DISPLAY

0008 EXECUTE 0

EXECUTE 1

EXECUTE 2

⋮

EXECUTE 9

TYPICAL SUBROUTINE

LBL 8 B Y SPC N 0 SPC

M E A N S PF RSET

PROGRAMMING HINTS

For those who have the optional printer, it is possible to write programs that prompt the user for input. Let us go back to the Ballistics program that required that 4 values be entered in order to begin. As you may recall, the Velocity was in K_0, the Distance to the wall was in K_1, the Height of the wall in K_2 and the thickness of the wall in K_3.

Using the printer and the alphabetic capability of the calculator you can initialize these values interactively. After you have keyed in the Ballistics program from pages 2-73 to 2-74, Key in the interactive initialization routine with the following keys.

Move to end of Ballistic Program and enter Learn.

[LABEL ■] [DEG RAD] [PAPER FEED]

PRESS [HOLD FOR ALPHA] ENTER INITIAL RELEASE [HOLD FOR ALPHA] [PAPER FEED]

PRESS [HOLD FOR ALPHA] VELOCITY IN RELEASE [HOLD FOR ALPHA] [PAPER FEED]

PRESS [HOLD FOR ALPHA] FEET PER SECOND RELEASE [HOLD FOR ALPHA]

[CLEAR] [STOP] [PRINT DSPLY] [=] [K] [0] [PAPER FEED]

PRESS [HOLD FOR ALPHA] DISTANCE TO WALL RELEASE [HOLD FOR ALPHA] [PAPER FEED]

PRESS [HOLD FOR ALPHA] IN FEET RELEASE [HOLD FOR ALPHA]

[CLEAR] [STOP] [PRINT DSPLY] [=] [K] [1] [PAPER FEED]

PRESS [HOLD FOR ALPHA] HEIGHT OF WALL RELEASE [HOLD FOR ALPHA] [PAPER FEED]

PRESS HOLD FOR ALPHA IN FEET RELEASE HOLD FOR ALPHA

CLEAR STOP PRINT DSPLY = K 2 PAPER FEED

PRESS HOLD FOR ALPHA THICKNESS OF RELEASE HOLD FOR ALPHA PAPER FEED

PRESS HOLD FOR ALPHA WALL IN FEET RELEASE HOLD FOR ALPHA

CLEAR STOP PRINT DSPLY = K 3 PAPER FEED PAPER FEED START

Exit the LEARN mode.

Now to initialize

PRESS

EXECUTE DEG RAD

As each question is asked enter the value and press CONT.

DEBUG & EDIT

DEBUGGING

Realize that the calculator can only perform the execution of a program according to the instructions contained within the program. This means that when the calculator is not performing an execution as you think it should, an error probably exists in the program. You must then proceed to find (debug) and correct (edit) this error.

Suppose that you have written a program that does not work. How should you proceed to find and correct the error? First, check the calculator memory to see if it contains the program steps you think should be there; you must obtain a *listing* of your program. Address the portion of memory that contains the program steps in question (press GO TO d d d d). Now press LIST.

LIST

The display will sequentially list all memory locations and the octal keycodes of the program steps they contain. The format of the list is given below. Each display lasts for approximately one second before it is replaced by the next.

0212 0124 0

Location 0212 contains x^2.

The current file is 0

With a printer, simultaneous with the display readout, LIST will result in a printout showing both the location and mnemonic of each stored program step. The format is given below in a sample listing that shows the program steps x^2 + K 0 = stored in memory beginning at location 0212.

```
0212  X↑2
0213  +
0214  K.
0215  0
0216  =
```

The listing will continue until interrupted by a manual STOP, or until the end of memory is reached. Upon reaching the end of memory, an (E 0) error message will appear in the display. Reset this error message by pressing the LEARN key. With a listing thus obtained, compare the listed memory contents against your own listing. To correct any discrepancies, enter the Learn mode at the proper location, press the desired key, and exit the Learn mode; the entered keystroke replaces the erroneous one. You may then proceed to find other errors or run the corrected program, whichever is appropriate.

Instead of obtaining a complete list of a program, you may decide that discrete portions of the program need close examination. Address the questionable program step as before (GO TO d d d d) and press DISPLAY PROGRAM; the addressed location and its stored program step are revealed in the display. Press STP→ to reveal subsequent locations. If you do not know or remember the keystroke that corresponds to the displayed octal code, press PRINT DISPLAY; the location and mnemonic of the displayed program step will be printed. To illustrate, suppose we have an x^2 stored in memory at location 0212;

DISPLAY PROGRAM

DEBUG & EDIT

PRESS GO TO DISPLAY 0 2 1 2 DISPLAY PROGRAM

Display: 0212 124 0

PRESS PRINT DSPLY

Printout: 0212 X↑2 F0

Again, when you find an erroneous keystroke, enter the Learn mode at the appropriate location, enter the desired keystroke, and exit the Learn mode.

Another method of program error detection is *stepwise* execution, in which program steps are executed in a step-by-step manner. As each program step is executed, the *result* of each execution appears in the display, just as if the stored program steps were manually entered from the keyboard. Stepwise execution requires that you understand roughly what the approximate result of each program step should be, and thus, when an unexpected result appears in the display, indicating what might be an error, you can then inspect the suspicious program step by the methods previously discussed. During stepwise execution, a DISPLAY PROGRAM will reveal the next program step to be executed.

←STEP

To stepwise execute a stored program, address the beginning of the locations of interest (GO TO d d d d) and sequentially press STP→ the *result* of the execution of each stored program step is revealed in the display. If at any time you desire to inspect the *next* program step to be executed, press DISPLAY PROGRAM. A PRINT DISPLAY will then reveal, as before, the mnemonic of the stored program step. If you see during stepwise execution that an erroneous program step was executed, press LEARN and STP← to index the counter to the just-executed program step. Then enter the desired program step, and exit the Learn mode with a second LEARN keystroke. (The second LEARN sets the counter back to 0000.)

Note that during stepwise execution, whenever DISPLAY PROGRAM is in effect, although the display only reveals program information, execution results are not lost. That is, a second DISPLAY PROGRAM will put *current* results back into the display, in accordance with the program steps that were executed while the DISPLAY PROGRAM was in effect.

Stepwise execution of stored programs is probably the most useful method of debugging and is especially advantageous on programs with many branches. However, in programs with loops, stepwise execution may not be the best method of debugging. Here, it may be better to examine a listing of the program and carefully check it against a flow chart.

EDITING

Program editing refers to program revision and final preparation. For example, when we write a program, our primary concern is usually the basic computation and smooth program flow from start to finish. However, somewhere along the line, we typically find need to *insert* extra program steps that we overlooked the first time through; we also may find need to *delete* extraneous program steps. In addition, in order to make wise use of our time and that of the calculator, it is often advantageous to totally revise the organization of a program. For instance, an often-called subroutine buried deep in memory may be better placed near the beginning of memory in order to minimize search time during program execution.

Using the editing capabilities of the calculator, all of the above operations are easily accomplished. In order to illustrate these techniques let's try a sample problem. First enter the following program — it will put 0123456789 into the display.

DEBUG & EDIT

PRESS [RESET] [LEARN] — Enter the Learn mode at 0000

[CLEAR DSPLY] [1] [2] [3] [4] [5] [6]

[7] [8] [9] [STOP] — The program

[LEARN] — Exit the Learn mode

To run the program press START; the display will show 0123456789, as expected.

To list the program press RESET and LIST; since RESET directs the counter to 0000, that is where our list begins. (Press STOP to stop the list after the program has been listed.)

MEMORY LOCATION	DISPLAYED OCTAL CODES	WITH PRINTER
0000	040	CLDP
0001	061	1
0002	062	2
0003	063	3
0004	064	4
0005	065	5
0006	066	6
0007	067	7
0008	070	8
0009	071	9
0010	043	STOP

DEBUG & EDIT

Stepwise execution is accomplished with a RESET and STP→, STP→, STP→, etc. Note that each STP→ results in the entry of the executed keystroke into the display; the eleventh STP→ executes the STOP and causes illumination of the *STOP* light. During stepwise execution, press DISPLAY PROGRAM at any time to display the octal keycode of the next step to be executed.

INSERT

[INSERT]

The INSERT key is non-programmable. In the Idle mode, this keystroke will cause an (E 4) error message (requires Learn mode). In the Learn mode, any keystroke that follows INSERT will be inserted into the program at the location currently displayed and the currently displayed program step and all program steps following the displayed step are displaced in memory by one location. To insert several steps, press INSERT the appropriate number of times and then press the corresponding keystrokes for the steps you want to insert.

To illustrate the above, let's insert a 3 at location 0003 in the example program.

PRESS	[GO TO] [0] [0] [0] [3] [LEARN]	Enter the Learn mode at the designated location
	[INSERT] [3]	Insert a 3 at step 0003
	[LEARN]	Exit the Learn mode

DEBUG & EDIT

The program now looks like this:

0000	CLDP	
0001	1	
0002	2	
0003	3	This is the inserted program step
0004	3	
0005	4	
0006	5	
0007	6	
0008	7	
0009	8	
0010	9	
0011	STOP	

To run the program press START; the display shows 1233456789. Note the difference between this display and the previous one — the extra 3 is a result of the inserted program step. Note also that the new program contains one more program step and every program step after step 0003 has changed one location downward in our listing.

DELETE

DELETE

Like INSERT, DELETE is non-programmable and its use in the Idle mode will result in an (E 4) error message. When DELETE is used in the Learn mode, the keystroke that is presently displayed will be removed from the program and all program steps following it will be moved up in memory one location.

To illustrate this, let's remove the extra 3 from the example program.

PRESS	GO TO 0 0 0 3 LEARN	Enter the Learn mode at 0003
	DELETE	Delete program step 0003
	LEARN	Exit the Learn mode

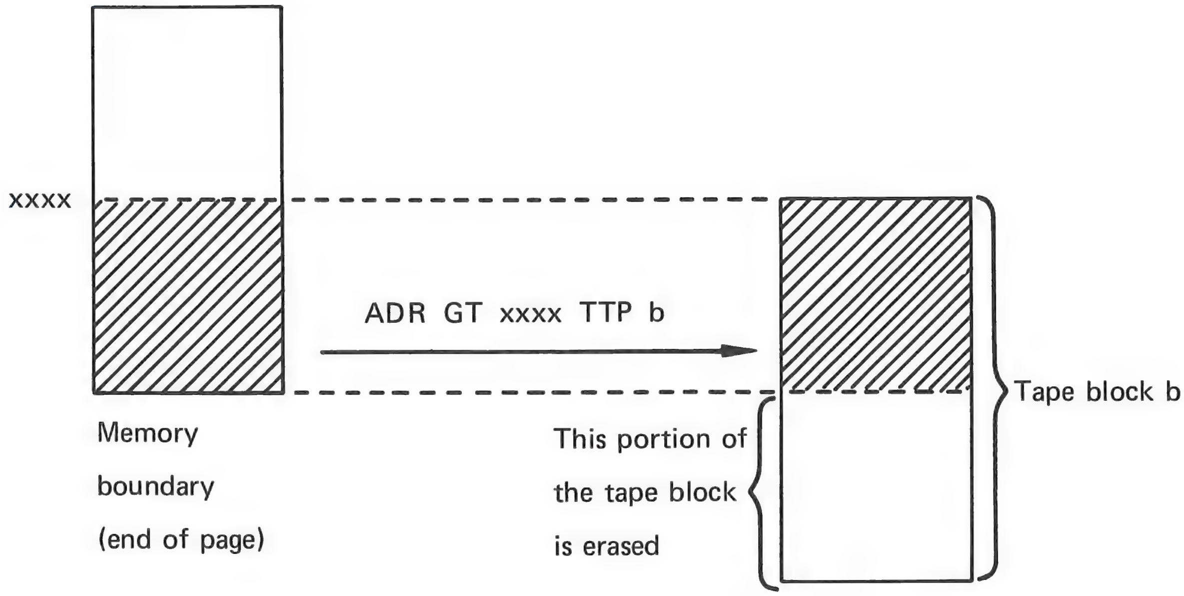
The program is thus restored to its original form. To run it, press START; the display shows 0123456789, as before. A list of the program will reveal that the extraneous 3 is deleted.

PROGRAM EDITING WITH THE MAG-TAPE

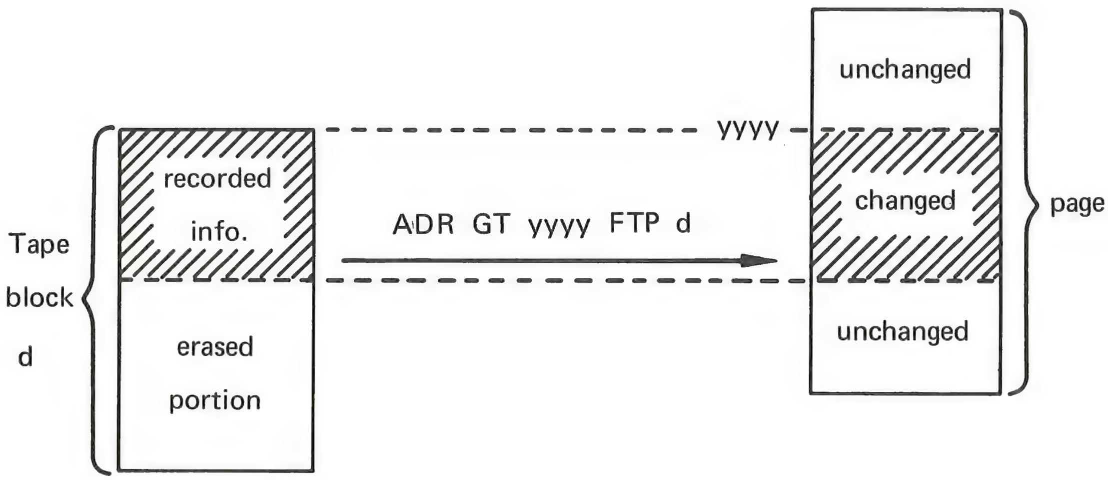
Often, insertion and deletion of program segments is necessary. In these circumstances the mag-tape can be used to edit programs. The following discussions depict the methods by which this is achieved.

UTILIZING MEMORY BOUNDARIES

When a memory boundary (end of page, file, or memory) is encountered while program steps or R-registers are transferred to a tape block, the portion of the tape block following the boundary is erased.



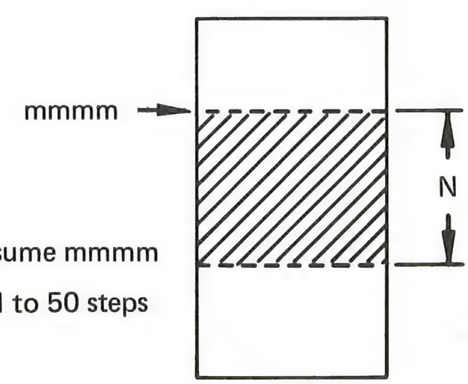
Later, when the contents of the tape block are loaded into the calculator memory, only the recorded information is transferred; the erased portion of the tape will *not* change the rest of the page or file.



This technique allows you to transfer a given program segment to any location within a page of memory without altering those steps which follow the transferred program segment.

To arrange a program segment on tape , consider the following example.

A program segment begins at memory location mmmm and is N steps long.

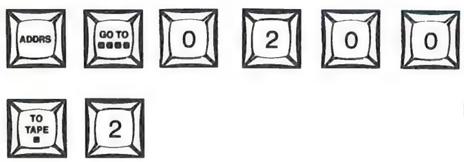


For this EXAMPLE assume mmmm is 0200 and N is equal to 50 steps

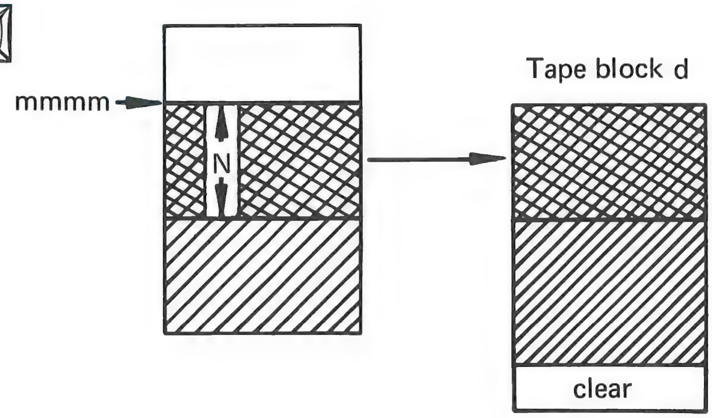
DEBUG & EDIT

(1) Transfer the program segment to tape so that the first step recorded on the tape is the first step of the program segment.

ADR GT mmmm TTP d

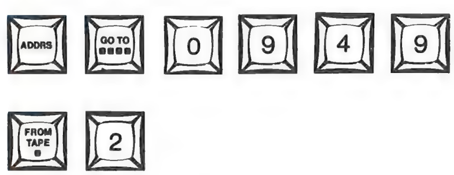


tape block 2 is arbitrary



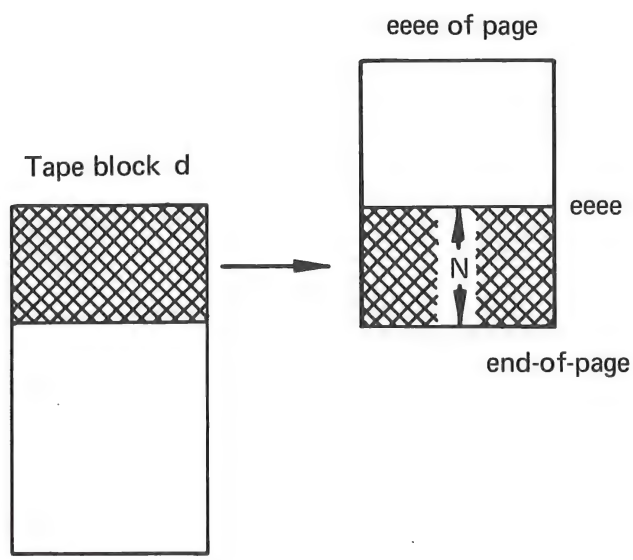
(2) Transfer the program segment back into the calculator memory so that the last step of the segment is the last step in a memory page.

ADR GT eeee FTP d



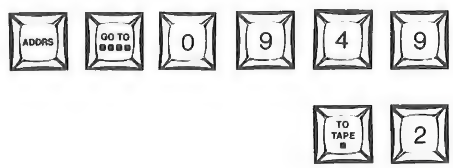
end-of-page equals 0999

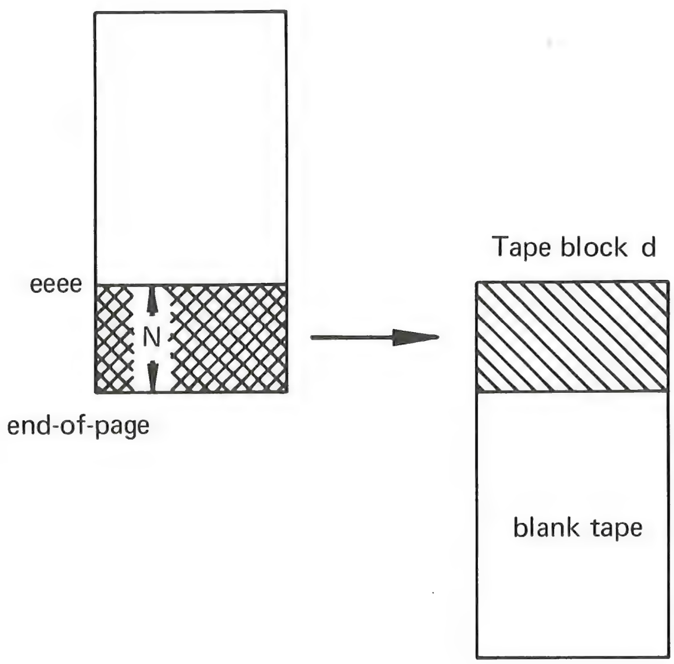
eeee equals end-of-page address minus N, where N is the number of steps in the program segment



(3) Transfer the program segment back to the tape so that only the program segment is transferred to the tape and the rest of the tape is erased.

ADR	GT	eeee	TTP	d



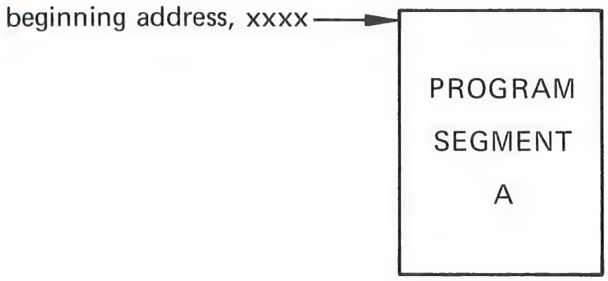


The program segment is now recorded on the tape and may be transferred into the calculator memory so that only N steps of the memory will be altered. Notice that if subroutines are stored on tape in this manner, it becomes a simple process to build a program from these subroutines.

Often, you may be concerned only with the arrangement of program segments and not concerned with what is contained in the program memory following the newly arranged program segments. In this case, it is not necessary to isolate the program segments on a single tape block. The following examples illustrate the techniques used.

MOVE A PROGRAM SEGMENT IN MEMORY:

A program segment, A, with beginning address, xxxx may be moved in memory so that it begins at another location, yyyy.



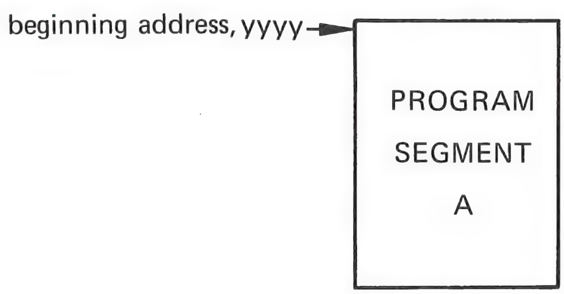
Record program segment A on tape block 1.

ADR GT xxxx TTP 1

(Note: All the program steps from address X to the end of the page will be recorded on block 1.)

Enter contents of tape block 1 into the calculator memory starting at location yyyy.

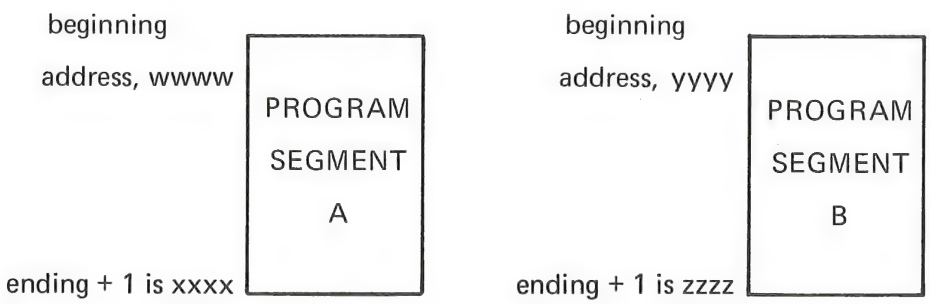
ADR GT yyyy FTP 1



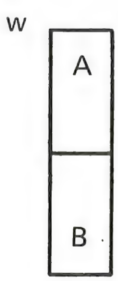
Result of above: Program segment A is now in memory starting at beginning address, yyyy.

LINKING SEPARATED PROGRAM SEGMENTS:

Suppose we have two program segments, A and B, separated from each other by unwanted or unused program steps. Suppose further that we wish to *edit* the program so that A and B are adjacent in memory.

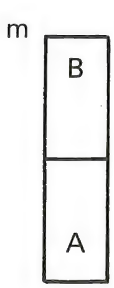


The wwww and yyyy are the starting addresses of program segments A and B respectively. xxxx and zzzz the ending addresses *plus one*. (wwww < xxxx < yyyy < zzzz).



To LINK B with A:

ADDRS GO TO yyyy TO TAPE Ø	Record B on tape block 0
ADDRS GO TO xxxx FROM TAPE Ø	Enter B into memory starting at X



To LINK A with B:

ADDRS GO TO wwww TO TAPE 1	Record A on block 1
ADDRS GO TO yyyy TO TAPE 2	Record B on block 2

Now enter the taped program segments into the calculator memory starting at location mmmm, where mmmm is any starting address.

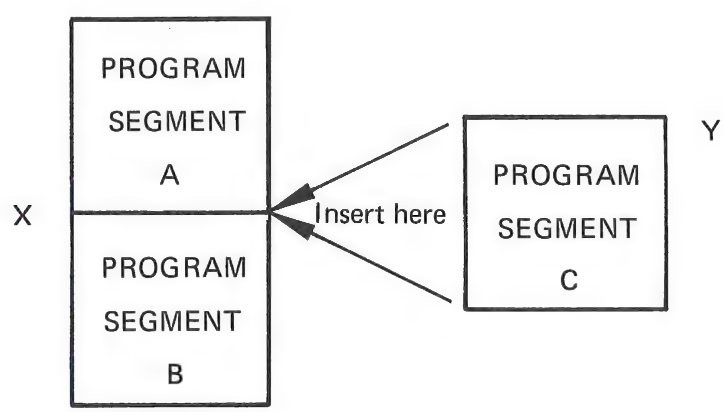
[ADDRS] [GO TO] mmmm [FROM TAPE] [2] — Enter B into memory starting at location mmmm.

Now, find the end of the new B program segment. Here we will call it nnnn.

[ADDRS] [GO TO] nnnn [FROM TAPE] [1] — Enter A into memory following the last step of the new, relocated B segment.

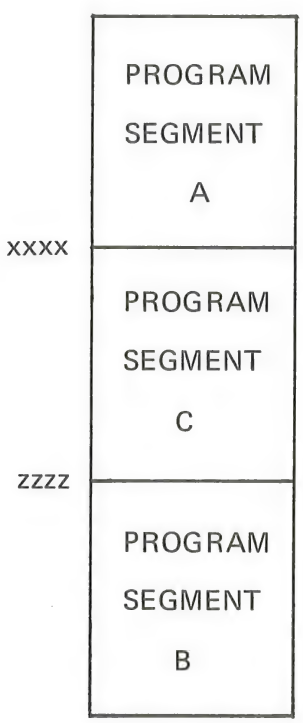
INSERTING A PROGRAM SEGMENT BETWEEN TWO OTHERS

Suppose that A and B are adjacent in memory and we wish to insert another program segment, C, in between A and B.



1. [ADDRS] [GO TO] xxxx [TO TAPE] [1] — Record B on tape block 1.
2. [ADDRS] [GO TO] yyyy [TO TAPE] [2] — Record C on tape block 2 .
3. [ADDRS] [GO TO] xxxx [FROM TAPE] [2] — Enter C into memory starting at X.
4. [ADDRS] [GO TO] zzzz [FROM TAPE] [1] — Enter B into memory starting at the program step following the last step in the just-entered C; here we call that Z.

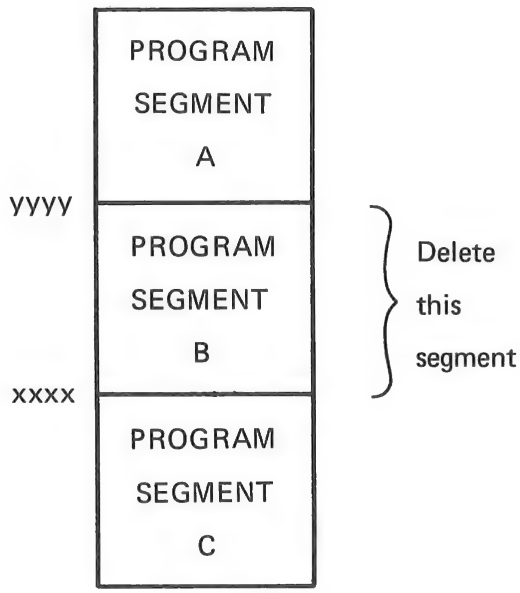
RESULT: As diagrammed below, program segment C is now inserted between program segments A and B.



DEBUG & EDIT

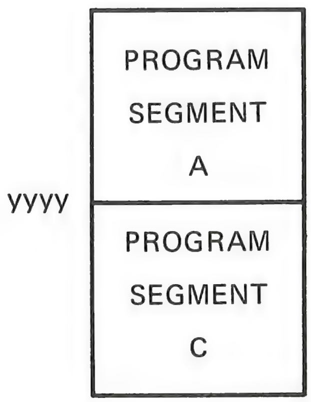
DELETING PROGRAM SEGMENTS WITH THE MAG-TAPE

Suppose that we have three adjacent program segments, A, B, and C, and that we wish to delete the B segment from the program.



ADDRS GO TO xxxx TO TAPE 1	Record C on tape block 1.
ADDRS GO TO yyyy FROM TAPE 1	Enter C into memory starting at yyyy.

RESULT:



Program segment B is deleted.

PROGRAMMING WITH A PRINTER

PROGRAMMING WITH THE PRINTER

The printer on your calculator may be used for several purposes. There are two main purposes: 1) The printer is used to list programs — both during and after program entry. 2) With the printer and its alpha capabilities, you can write interactive programs, an example of which might be directing the calculator to ask you for specific data inputs and print out titled solutions.

A programmed PRINT DISPLAY will cause the current contents of the display to be printed. All displays are printed as they appear with the following exceptions:

* Flashing displays are printed with a # between the mantissa and exponent. For example,

FLASHING	PRINTED
3.141592653	3.141592653#
1.234567890 –03	1.234567890#–03

* Leading zeros are suppressed from the printout and a display consisting of all zeros is printed as a single zero.

DISPLAY	PRINTED
000001.2303	1.2303
0000000000	0.

* When a PRINT DISPLAY is encountered by the calculator while a program is being executed and a DISPLAY PROGRAM is in effect, the location and mnemonic of the program step *following* the print command are printed. To illustrate, enter the following program:

PROGRAMMING WITH A PRINTER

RESET LEARN	Enter the Learn mode at 0000
CLEAR DSPLY 1 2 3 4	Enter 1234 into the display
PRINT DSPLY	Print the contents of the display
START	Loop back to 0000 and begin again
LEARN	Exit the Learn mode

A START will execute the program; press STOP to terminate. The printouts are given below.

NORMAL PRINTOUT	PRINTOUT WITH DISPLAY PROGRAM IN EFFECT
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0
1234.	0006 STRT F0

In the above, note the difference between the two printouts — when the DISPLAY PROGRAM is in effect, the step following the print command is printed; this is useful when tracing the locations of various print commands in lengthy programs.

NOTE: When running a program with DISPLAY PROGRAM in effect, the calculator will not print alpha messages.

PROGRAMMING WITH A PRINTER

PROGRAMMING ALPHA

Alpha printouts are used to format data entries and subsequent printed solutions. In order to enter alpha information into the calculator memory, understand the following:

* All available alpha characters, except digits and parenthesis, are printed in the blue squares adjacent to their respective keys. The numeric keys and parenthesis keys have no blue squares, but are used to print their corresponding alpha characters.

* Actuate alpha input by depressing the blue HOLD FOR ALPHA key, HFA, *during* any alpha keystrokes.

* The paper will contain sixteen characters on a line.

* Alpha information in a program is printed when a total of sixteen characters have been entered, *or* when the alpha program steps are followed by any programmed non-alpha command. To illustrate:

This program

```
0000   CLDP
0001      A
0002      B
0003      C
0004      D
0005      E
0006      F
0007      G
0008      H
0009      I
0010      J
0011      K
0012      L
0013      M
0014      N
0015      O
0016      P
0017      Q
0018      R
0019      S
0020      T
```

```
0021      U
0022      V
0023      W
0024      X
0025      Y
0026      Z
0027   STOP
```

Results in this printout

```
ABCDEFGHIJKLMNOP
QRSTUVWXYZ
```

The first sixteen alpha characters are printed when the line is full.

The other ten characters are printed as a result of the STOP command, as it is the first keystroke following the alpha.

PROGRAMMING WITH A PRINTER

* A PRINT DISPLAY following alpha instructions will cause both the alpha information and the contents of the display to be printed, alpha first, and display second, For example, the program steps, CD π L O O K PRINT DISPLAY will cause the following printout:

```
LOOK
 3.141592653
```

The same applies in the Idle mode; that is, the above instructions in the Idle mode will cause the same printout.

A good practice program with the alpha keys is to program in your own personal address label. The one shown here may be used as a guide.

```
0000       T
0001       E
0002       K
0003       T
0004       R
0005       O
0006       N
0007       I
0008       X
0009
0010       I
0011       N
0012       C
0013       .
0014    PAPR
0015       P
0016       .
0017       O
0018       .
0019
0020       B
0021       O
0022       X
0023
0024       5
0025       0
0026       0
0027    PAPR
0028       B
0029       E
0030       A
0031       V
0032       E
0033       R
```

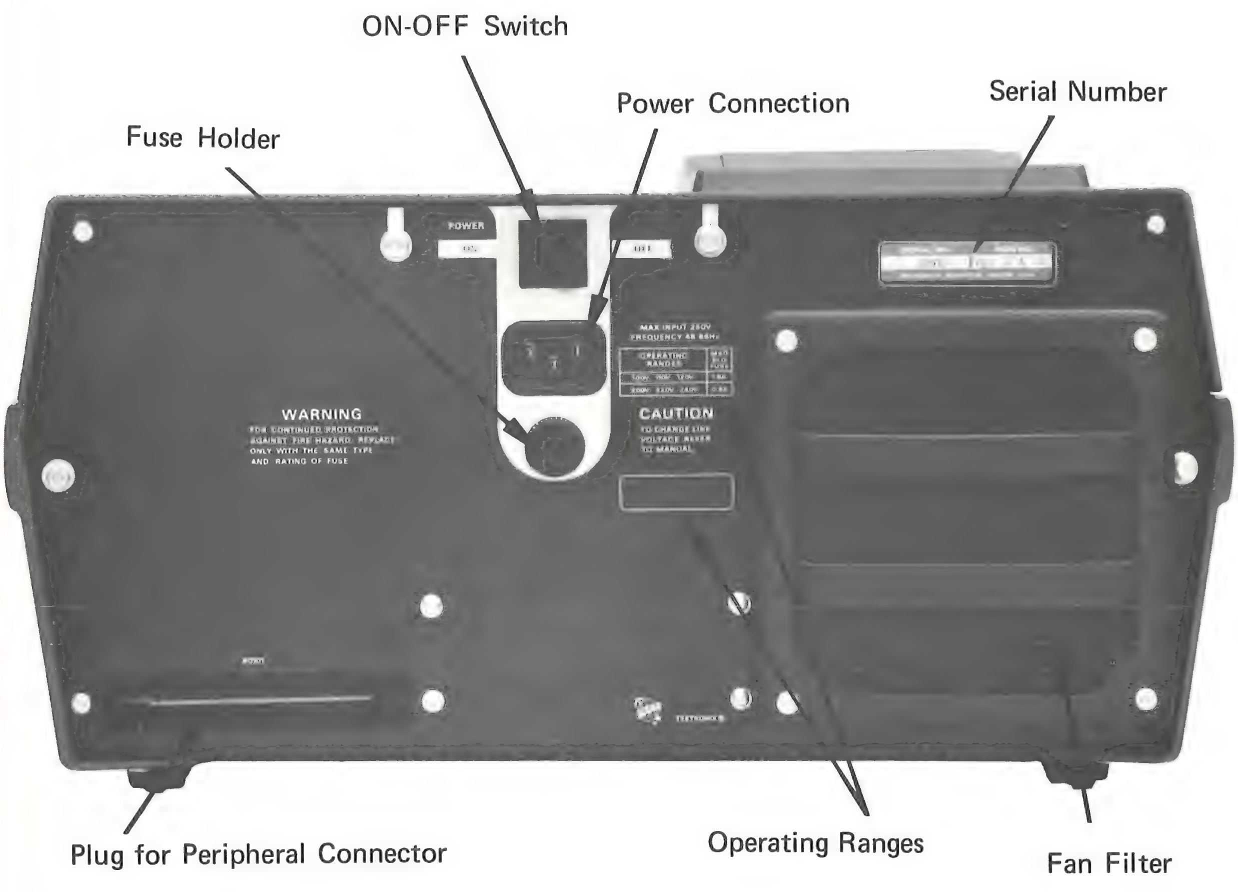
```
0034       T
0035       O
0036       N
0037       ,
0038
0039       O
0040       R
0041       E
0042       .
0043    PAPR
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054       9
0055       7
0056       0
0057       0
0058       5
0059    PAPR
0060    PAPR
0061    PAPR
0062    PAPR
0063    PAPR
0064    STRT
0065    NULL
```

```
TEKTRONIX INC.
P.O. BOX 500
BEAVERTON, ORE.
          97005


TEKTRONIX INC.
P.O. BOX 500
BEAVERTON, ORE.
          97005


TEKTRONIX INC.
P.O. BOX 500
BEAVERTON, ORE.
          97005
```

INITIAL OPERATION

BACK PANEL

The back panel contains the on-off switch, power cord plug-in, fuse holder, information on operating power ranges, the serial number, the fan filter, and a plug for the peripheral connector.



LINE VOLTAGE

Line voltage is single phase, grounded neutral with nominal voltages; 100, 110, 120, 220, or 240 AC volts. Satisfactory calculator operation is achieved within a ±10% range of the above voltages. An indicator on the back panel shows the rated voltage set at the factory.

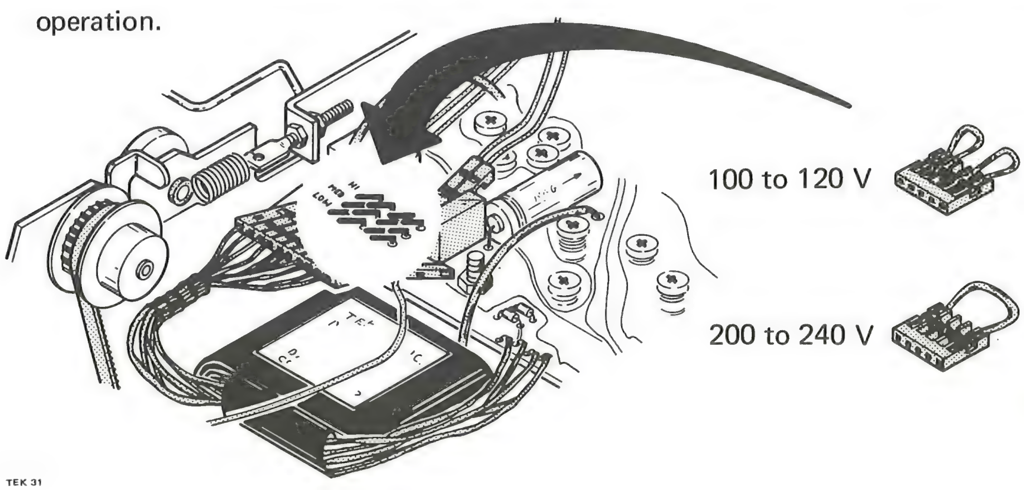
Before you connect the calculator to a power source, verify that the rated voltage matches that of your source —

CHANGING RATED VOLTAGE

To change the rated voltage to 100, 110, 120, 200, 210, or 240, follow the steps outlined below:

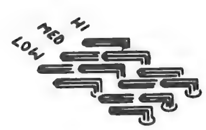
1. Turn off power.
2. Remove the lid as follows: loosen the two screws on the back panel, then bend the vertical surfaces of the lid slightly as shown below to release the lid. Lift the lid straight up and set it aside.

3. The power supply board has three rows of five pins each. The pins are jumpered by a connecting block which contains jumper wires. Remove the block by pulling it out. Change the jumper wires as indicated below for 100 to 120 or 200 to 240 volt operation.



INITIAL OPERATION

4. Place the connector back onto the pins according to the desired voltage rating.



5. Re-install lid and change fuse.

6. Indicate new voltage rating on back panel.

FUSES

The fuse holder is located on the rear panel. For 100 to 120 VAC operation, use 1.6 amp slow-blow fuse. For 200 to 240 VAC operation use 1/2 amp slow-blow fuse.

If a new fuse blows immediately on replacement, check FUSES above to verify that you have installed the correct fuse for the rated voltage. If the correct fuse was blown a second time, DO NOT replace it with one of higher current rating — this may lead to damage of the calculator. Instead, call your Tektronix Field Office for assistance.

INITIAL OPERATION

LINE FREQUENCY

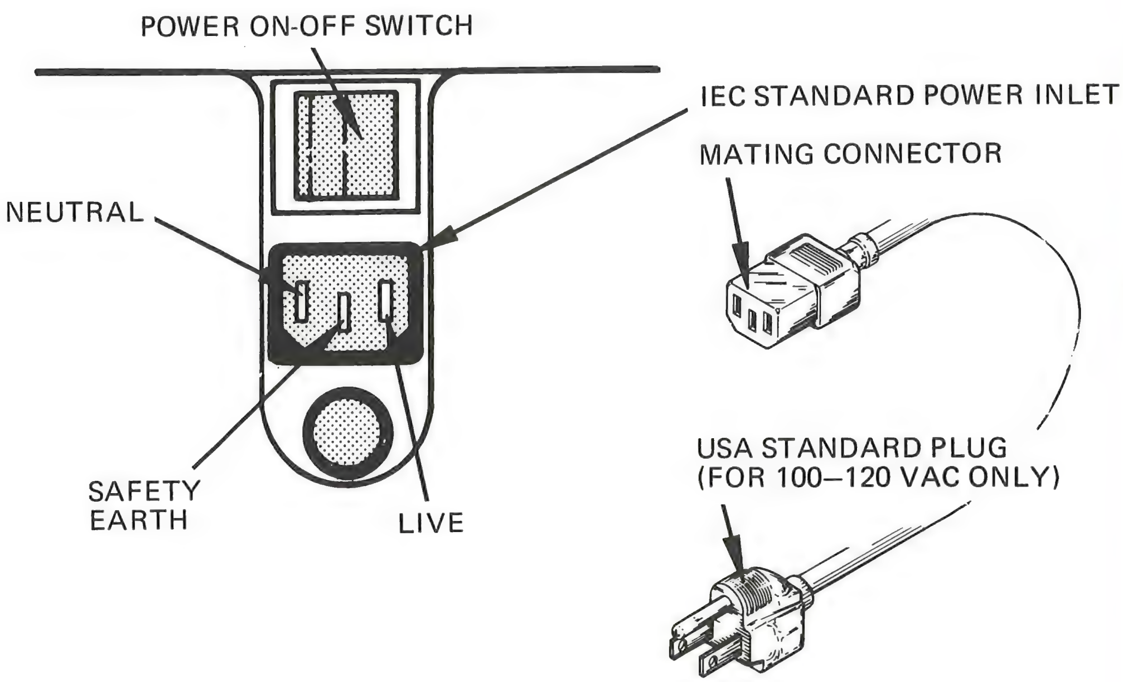
Satisfactory operation is achieved with line frequencies from 48 to 62 Hz.

POWER REQUIREMENT

The power requirement is a maximum of 100 voltamperes.

GROUNDING REQUIREMENTS

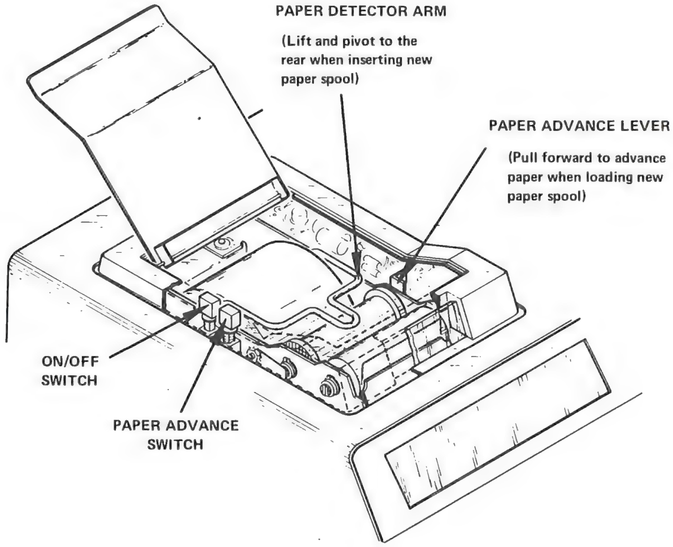
World safety standards suggest that frame, chassis, and keyboard cabinet be grounded (earthed) to minimize electrical shock hazards. The calculator is equipped with a three conductor power cord set that will efficiently ground it when used in conjunction with an appropriately grounded power source receptacle.

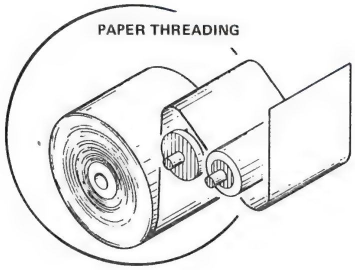


INITIAL OPERATION

LOADING PRINTER PAPER

If your calculator is equipped with a printer, load it with paper according to the following procedure:





1. Raise the Printer Access Cover.

2. Remove any core remaining from an old roll or lift out any remanant of a remaining roll (tear cleanly and feed out the scrap by pressing the PAPER button on the printer).

3. Unwind soiled paper from new roll and cut off or tear cleanly.

4. Insert the new roll according to the diagram on the underside of the Printer Access Cover.

5. Close the access cover.

Standard Accessories:

one power cord set

one verification tape cartridge

one blank tape cartridge

1.6 AMP slow-blow fuses

0.8 AMP slow-blow fuses

User definable cards

Optional Accessories:

handle

ROM pack

five rolls printer paper

GENERAL INFORMATION

OPERATING CHARACTERISTICS

This chapter is devoted to discussions giving general information about the calculator and its operating characteristics, especially those that are not unique to any particular key, but are characteristic to the calculator as a whole. For the most part, detailed descriptions of individual keys are contained in preceding sections — here we will elaborate on particular key operations only to illustrate generalities of operation.

KEYBOARD ARRANGEMENT

The calculator keyboard is divided into two major sections: one for basic arithmetic operations and the other for programming operations.

Contained in the major sections are keys that control the mag-tape mechanism, thermal printer, and remote peripheral devices.

KEYBOARD COLOR CODING

The keys on the keyboard are color coded for quick identification. Adjacent to the various keys, the blue tints on the keyboard designate the alpha characters that are available for printout when these keys are used in conjunction with the blue HOLD FOR ALPHA key.

STATUS INDICATORS

The calculator display includes a group of six status indicators which, when illuminated, indicate current machine status or operating mode.

RAD	DEG	LEARN	BUSY	STOP	ADDR INCOMP

Ⓐ TEK 31

OPERATING CHARACTERISTICS

RAD and DEG indicate current trigonometric operating mode. RAD indicates radian operation, and DEG indicates degree operation. According to the status indicated, the calculator will take displayed numbers as *either* being in degrees *or* radians whenever any of the trigonometric keys are used. Change the trigonometric operating mode by pressing the D/R key. A CLEAR will always revert the trigonometric status to RAD.

LEARN indicates that the calculator is in the Learn mode; the program memory is open to accept and retain sequential entry of programmable keystrokes. In the Learn mode, the display consists of three numbers: the first number indicates the currently indexed memory location, the second number is the octal keycode of the keystroke stored at that location, and the third number, a single digit, indicates the presently accessed R-register file.

BUSY indicates that the calculator is engaged in one of the following activities:

- Program execution
- Printing
- Computing an arithmetic operation
- Transferring information to or from mag-tape

Exit from the Busy mode is achieved when the current activity is completed. During certain times in tape transfers and at all times during program execution, exit from the Busy mode may be accomplished by pressing STOP.

STOP indicates that a programmed STOP command has been executed, or that an activity, as discussed above, has been interrupted by a manual STOP keystroke.

OPERATING CHARACTERISTICS

ADDR INCOMP indicates that the calculator is waiting for memory location address digits, data register (K or R) address digits, a subroutine lable, a file number, a tape block number, or a remote peripheral address. The keys that require addresses are so indicated by the appropriate number of embossed white squares. (For instance, the K key requires a single digit address. This is indicated by one square on the key.)

Normally, when the calculator is not in the Busy or Learn modes (*BUSY* and *LEARN* lights out) the calculator is said to be in an Idle mode. In the Idle mode, the keyboard is fully functional. In the Busy mode, the keyboard is not functional and the *only* key that has any effect is the STOP key. In the Learn mode, all *programmable* keystrokes are entered into the program memory.

RANGE OF OPERATION

The operating range of the calculator is essentially from zero to $\pm 9.999999999 \times 10^{99}$. This is best expressed on the number line as,

$-x$ ←—— 0 ——→ $+x$

$-9.999999999 \times 10^{99}$ to -1×10^{-99} | $+1 \times 10^{-99}$ to $9.999999999 \times 10^{99}$

Symbolically, this range may be expressed as,

$$+1 \times 10^{-99} \leqslant |x| \leqslant 9.999999999 \times 10^{99} \text{ , and } x = 0$$

Operations resulting in results outside of this range will switch the calculator into what is called an *overrange* condition, which is indicated by a flashing display.

FLASHING DISPLAY AS ERROR INDICATION

A moment after turn-on, the calculator display will come on and flash, on and off about twice per second. To stop this flashing, press the CLEAR key. The flashing display is the principal mathematical operations error indicator and also occurs following any power interruption, indicating memory erasure. The flashing display is used to indicate that an illegal mathematical operation has been performed, or that the result of an operation exceeds the calculator operating range, as outlined above. Forbidden operations include the square root and log of a negative number, division by zero, and trigonometric functions outside the range of allowable arguments. A print command executed during a flashing display will cause a # to be printed between the mantissa and exponent.

It is noteworthy to mention here that a flashing display is an error message *only*. Given a flashing display, you may choose on one hand to acknowledge and CLEAR the flashing display, or you may choose to ignore the flashing display and continue your computations. In this case, the calculator will perform correctly in *subsequent* commands. The flashing display can only be returned to its former appearance by a CLEAR.

In some situations you may wish to deliberately create a flashing display to indicate that a certain condition exists in a program. The simplest way to do this is with a CD and 1/x (divide by zero). A flashing display of all zeros is achieved by following the above with another CD.

OPERATING CHARACTERISTICS

ERROR MESSAGES

Aside from overrange and illegal operations, which are indicated by flashing displays, other errors on the calculator are indicated by a series of ten displayed error messages, (E 0 through E 9). These error messages appear in the exponent portion of the display whenever the calculator determines that it has received erroneous keystrokes or when certain conditions exist that are important for you to know. The error messages are fully described in the appendix devoted to this topic. Briefly, they are as follows:

E 0 End of memory
E 1 No such step
E 2 No such register
E 3 Requires a digit
E 4 Requires Learn mode
E 5 No such label
E 6 Illegal code in memory
E 7 No cartridge, no paper, or no such tape address
E 8 Write protected cartridge
E 9 Bit error

DISPLAY AND DATA FORMATS

The calculator will accept input data in a variety of notations. Each of the following formats is equally acceptable and may be intermixed at will.

DATA FORMATS

DECIMAL or ORDINARY	100	–512	.0001	243.123
SCIENTIFIC	1×10^{2}	-5.12×10^{2}	1×10^{-4}	2.43123×10^{2}
MIXED	10×10^{1}	-51.2×10^{1}	$.01 \times 10^{-2}$	$.00243123 \times 10^{5}$

OPERATING CHARACTERISTICS

The calculator can display a maximum of twelve digits, a decimal point, and two signs. In ordinary or decimal notation, the calculator displays a ten digit mantissa with sign and a decimal point. In scientific notation, the display consists of a ten digit mantissa with a decimal point immediately to the right of the first significant digit, and a two digit exponent with sign.

When scientific notation has not specifically been called for (by using other notations), results of calculator operations are displayed in ordinary decimal notation when they are in the range of fixed point displays:

$$1 \times 10^{-10} < |x| \leqslant 1 \times 10^{+9} \qquad \text{(range of decimal notation)}$$

When any calculation exceeds the above range, the calculator will switch the display to scientific notation; it will revert to ordinary notation whenever the range of the display returns to the range of ordinary notation given above — *unless* scientific notation has been specified by entering data in scientific notation.

OPERATING CHARACTERISTICS

GUARD DIGITS

Both data and internal results of calculations are stored in the calculator to *twelve* decimal places, but are displayed to *ten* decimal places. The remaining two digits are called the guard digits — they function to maintain ten digit accuracy through successive calculations and provide for automatic rounding of the least significant displayed digit.

To demonstrate the above, divide 5 by 3 and then multiply the result by 3. After the division the result is 1.66666667, after the multiplication the result is 5, as expected. Now repeat the calculation but destroy the guard digits by pressing +/— twice after the division, then do the multiplication. The result is 5.000000001. The reason is that when we destroyed the guard digits, our accuracy suffered and the error of 0.000000001 is a result.

The guard digits are not normally displayed, but if you wish to examine them, simply subtract the two most significant digits from any display. For example, let's examine the guard digits on $\sqrt{3}$.

PRESS	[3] [$\sqrt{x}$]	Display: 1.732050808
	[−] [1] [.] [7] [)]	Display: 3.205080756 −02

The guard digits are 56. In the first display, 756 is rounded to 8.

OPERATING CHARACTERISTICS

In certain calculator operations the rounding action of the guard digits may seem to cause incorrect or anomalous displays. For example, if you compute 2^{10}, the display will yield 1024, as expected. But if you take the integer value of the display, you get 1023. If you examine the guard digits in the first display you will find that the 1024 is actually 1023.99999989 rounded up.

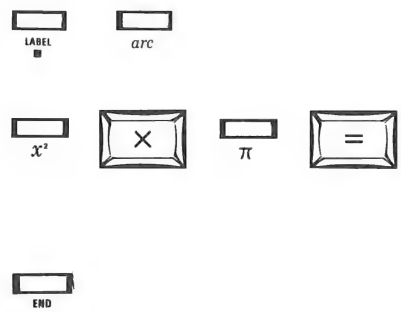
When programming certain kinds of trigonometric calculations or logic operations, you may at times wish to discard or kill the guard digits. To do this PRESS +/— twice; once to Kill the guard digits, and a second time to restore the original sign of the display. Alternate to this, keying in a O will accomplish the same purpose.

OPERATING CHARACTERISTICS

USING THE OVERLAY

The overlay furnishes you with the ability to *redefine* the function of each of the keys on the lower-left quadrant of the keyboard. For instance, suppose you have occasion to compute the area of a circle, given the radius in the display. Normally, you would press x^2 X π to arrive at the desired result in four keystrokes. However, by utilizing the overlay, this computation could be accomplished in a single keystroke. To do this, we would program the following subroutine into the calculator memory:

LABEL



This keystroke identifies the following keystrokes as a subroutine. arc is the label here, but any other of the 24 keys in the quadrant would suffice.

These keystrokes compute the area of the circle from the radius given in the display.

The END signals the calculator that the computation is complete.

On the overlay opposite the arc key we would write some appropriate identification such as "AREA" to indicate the function of the subroutine that this key labels. Now, for all practical purposes, with the subroutine in the memory and the overlay installed, the arc key is an AREA key. In this manner you may use any and all of the 24 keys in the quadrant to freely define your own special purpose functions, each of which may be executed by pressing the key adjacent to the appropriate label. When the overlay is removed, the keys revert back to their original functions.

In large programs containing many subroutines, individual subroutines may be of themselves useful to you, independent of their functions in the main program. Thus, you may desire to call upon these subroutines to *individually* perform their designed purposes. This may be done provided that the subroutines you call from the keyboard contain END commands (as outlined previously), and the overlay is labeled appropriately with the *purpose* of each utilized subroutine. In this manner the overlay would contain information regarding the function of each subroutine in the program and you would thus instruct the calculator to perform the functions that *you* have defined on the overlay, as opposed to those printed on the keyboard. For instance, suppose that you have a program that contains several subroutines; one for initialization, one for computation, and one for plotting. Utilizing the overlay, it would be marked with words such as "INITIALIZE", "COMPUTE", and "PLOT" opposite the keys that serve as labels for these respective subroutines. Then, with the overlay in place, each of these functions could be executed by singular keystrokes that are identified on the overlay with clear intent.

VERIFICATION

If you feel that your calculator is not operating properly the verification tape cartridge supplied with your calculator may be used to verify operation. Insert the verification tape into the tape transport as illustrated below.

Press [RESET] [FROM TAPE] [0]

After the program has been transferred into the calculator memory, press START. The calculator will stop with a cleared display. Enter the total number of R-Registers in your calculator and press continue.

The calculator will stop several times with various numbers in the display. Follow the table below to verify the operation of your calculator. (NOTE: the display should not be flashing unless so indicated.)

DISPLAYED NUMBER	PRESS	CONTINUE
1	"	"
2	"	"
3	"	"
4	"	"
5	"	"
6	"	"
7 FLASHING	"	"
8 FLASHING	"	"
9 FLASHING	"	"
10 FLASHING	"	"
11 FLASHING	"	"

VERIFICATION

After pressing CONTINUE the busy light will come on while the calculator loads the next set of program steps from the tape. After this search is completed, the program will automatically resume.

	PRESS CONTINUE
12 FLASHING	" "
13 FLASHING	" "
14	" "
15	" "
16	" "
17	" "
18	" "
19	" "
20	" "

After pressing CONTINUE the calculator will again search for the next series of program steps and the program will automatically resume.

21	PRESS CONTINUE
22	PRESS CONTINUE

At this point, if you have the printer option installed and turned on, the following will be printed: ??????????

```
ABCDEFGHIJKLM
NOPQRSTUVWXYZ

0123456789

.@(\)/*-+=
$ %;!:#&↑!
<?>[′"]_

 3.141592653

 1234567890.E+12
 1.234567890E-03
```

If the program does not run as outlined, contact your local Tektronix Field Office.

After performing the above verification procedure,

PRESS CLEAR REMOTE 2 2 The display should remain cleared.

TEK 31

REV. JAN 1974

ERROR MESSAGES

ERROR MESSAGES

APPENDICES

ERROR MESSAGES

Any error message may be reset by pressing LEARN twice in succession.

E0 End of memory; the program being executed, written or loaded has exceeded the number of program steps available in the memory of the calculator. (Reset by pressing RESET.)

E1 No such step; a branching command addresses a non-existent address. (Reset by pressing CLEAR or RESET.)

E2 No such register or file; a non-existent register or file has been called. (Reset by pressing CLEAR or RESET.)

E3 Requires a digit; GO TO, R■■, R■■■, K, FTP, TTP, and REMOTE all require a specific number of digits following them. The E 3 error message will appear whenever a non-digit key is pressed before the appropriate number of digits have followed the above keystrokes. (Reset by entering a digit or pressing LEARN twice.)

E4 Requires Learn mode; STEP, INSERT, and DELETE can only be used in the Learn mode. (Reset by pressing RESET or CLEAR.)

E5 No such label; a subroutine has been called that does not exist in memory. (Reset by pressing CLEAR or RESET.)

E6 Illegal code in memory; an illegal code such as LEARN or STEP is stored in memory. (Reset by pressing LEARN twice.)

ERROR MESSAGES

E7 No cartridge, no such tape block number, or no paper in the printer; results when there is no cartridge when a tape transfer is called, or when a non-existent block number is entered following TTP or FTP. This error message also occurs when the calculator is trying to print when there is no paper. (Reset by pressing CLEAR or RESET.)

E8 Write-protected; results when attempt is made to record a program and the rubber button on the front of the cartridge is removed, write-protecting the cartridge. Replace the button and repeat. (Reset by pressing CLEAR or RESET.)

E9 Bit error; during a tape transfer an unusual circumstance resulted in an incorrect number of bits in a character, and thus the code has not been stored or recorded correctly. (Reset by pressing RESET.)

KEYBOARD SYNOPSIS

KEYBOARD SYNOPSIS

Keystroke	Mnemonic	Pressed In Idle Mode	Executed In A Program	Learn Mode
STEP	STP →	Executes program step currently indexed, advance counter	Not programmable	Advances counter
STEP	STP ←	E4; requires Learn mode	Not programmable	Decrements counter
INSERT	NSRT	E4; requires Learn mode	Not programmable	Increments displayed step and all program steps after displayed step one location in memory. Next programmable keystroke is inserted at displayed location.
DELETE	DLT	E4; requires Learn mode	Not programmable	Deletes displayed step from program. Decrements all subsequent steps one location in memory.
LIST	LIST	Starts a program list starting at presently indexed location.	Not programmable	Same as Idle mode
DISPLAY PROGRAM	DSPG	Displays present location, keycode of program step stored at that location, and presently accessed file number.	Not programmable	Same as Idle mode
HOLD FOR ALPHA	HFA	Any keystroke entered while HFA is depressed is taken as an alpha keystroke	Not programmable	Same as Idle mode except it is used when storing alpha keystrokes in memory.
LEARN	LRN	Causes entry into Learn mode	Not programmable	Causes exit from Learn mode (entry into Idle mode)

KEYBOARD SYNOPSIS

Keystroke	Mnemonic	Pressed In Idle Mode	Executed In A Program	Learn Mode
EXECUTE	EXC	Causes execution of labeled subroutine when followed by labeling keystroke. END is recognized.	Same as Idle mode, except that END is not recognized.	Programmable
LABEL	LBL	When followed by labeling keystroke, counter indexes to first-step in corresponding subroutine	Indicates beginning of symbolically labeled subroutine	Programmable
END	END	No effect	When subroutine containing END is executed from Idle mode, END causes termination of execution. Otherwise, when subroutine is executed under program control, END is ignored.	Programmable
RETURN ADDRESS	RADR	Puts contents of address register into display	Same as Idle mode	Programmable
GO TO DISPLAY	GODP	Branch to displayed location (last four digits) and remain in Idle mode	Branch to displayed location and continue execution	Programmable
CLEAR FLAG	CLFG	Clears flag	Clears flag	Programmable
SET FLAG	SFG	Sets flag	Sets flag	Programmable
CLEAR R FILE	CFI	Clears all R-registers in file indicated in following keystroke	Same as Idle mode	Programmable
PAUSE	PAUS	No effect	Causes a pause in program execution. Pause is approximately one second in duration.	Programmable

KEYBOARD SYNOPSIS

Keystroke	Mnemonic	Pressed In Idle Mode	Executed In A Program	Learn Mode
$\geqslant 0$	IF$\geqslant$0	No effect	When condition is fulfilled, program steps following the IF statement are executed. Otherwise, the program branches to the next progammed CONT	Programmable
= 0	IF=0	No effect		Programmable
$<$ 0	IF$<$0	No effect		Programmable
FLASH	IFFL	No effect		Programmable
FLAG	IFFG	No effect		Programmable
RESET	RSET	Branch to 0000. Remains in Idle mode	Branch to 0000, exits Busy mode and enters Idle mode.	Programmable
START	STRT	Starts program execution at 0000.	Branch to 0000.	Programmable
ADDRS	ADR	Used when specifying Mag-tape transfers, always followed by GO TO or one of the R keys.	Same as Idle mode	Programmable
GO TO	GT	When followed by address digits, this is an unconditional branch command. Branch to specified location, remains in Idle mode.	Same as Idle mode except program execution is continued at specified location.	Programmable

KEYBOARD SYNOPSIS

Keystrokes	Mnemonic	Pressed In Idle Mode / Executed In A Program	Learn Mode
DEGRAD	D/R	Changes trigonometric operating mode from degrees to radians and vice-versa. Affects the trigonometric keys, ARC, TAN, COS, SIN.	Programmable
		Used also to convert displayed numbers from radians to their equivalent in degrees, and vice-versa.	Programmable
arc	arc	Used in conjunction with other trigonometric keys to find inverse arguments of displayed trigonometric variables.	Programmable
hyper	hyp	Used in conjunction with trigonometric keys when finding hyperbolic functions.	Programmable
tan x	tan	Tangent of the displayed angle.	Programmable
cos x	cos	Cosine of the displayed angle.	Programmable
sin x	sin	Sine of the displayed angle.	Programmable
x!		X Factorial of the integer of the displayed number	Programmable
Π_4	Π_4	Multiply K_4 by display and store the result in K_4	Programmable
Δ_3	Δ_3	Multiplies contents of K_3 register by –0.1 and stores result in K_3	Programmable
$_3\Sigma_2$	$_3\Sigma_2$	Adds contents of K_3 to K_2 and stores the result in K_2	Programmable
Σ_1	Σ_1	Adds display to K_1 and stores the result in K_1	Programmable
Σ_0	Σ_0	Adds display to K_0 and stores the result in K_0	Programmable
ln x	ln	Natural log of displayed number	Programmable
log x	log	Log, base ten, of displayed number	Programmable
int x	int	Integer value of displayed number	Programmable
$\sqrt{\Sigma^2}$	$\sqrt{\Sigma^2}$	Square root of the sum of the squares. Operates on sequentially entered numbers	Programmable
$\lvert x \rvert^a$	$\lvert x \rvert^a$	Raises display to a power subsequently entered.	Programmable
REMOTE	RMT	Used to address peripheral devices.	Programmable
e^x	e^x	e, the base of naperian logarithms (2.71 . . .) is raised to the displayed power and put into display.	Programmable

KEYBOARD SYNOPSIS

Keystrokes	Mnemonic	Pressed In Idle Mode	Executed In A Program	Learn Mode
R■■■	R■■■	Causes recall of stored contents of register specified by digit keystrokes that follow; the first digit indicates the file, and the second two indicate the R-register in the indicated file.		Programmable
R■■	R■■	Causes recall of stored contents of register specified by the digit keystrokes that follow. Recall is from *current* file. (Note: When an = precedes either of the above, the display is stored in the specified register.)		Programmable
TO TAPE	TTP	Used in mag-tape transfers of information *to* tape. This keystroke is followed by a single digit indicating the tape block to which the information is to be transferred. (0-5)		Programmable
FROM TAPE	FTP	Used when transferring information *from* mag-tape to the calculator memory. Followed by single digit keystroke indicating the tape block from which the information is to be taken. (0-5)		Programmable
STOP	STOP	No effect except to illuminate the *STOP* light under the display	Interrupts program execution. STOP is the only key that is functional while the calculator is in the Busy mode.	Programmable
CONT	CONT	This is an execution command directing the calculator to begin execution of the program at wherever the counter is presently indexed.	Performs no operation.	Programmable

APPENDICES

KEYBOARD SYNOPSIS

Keystrokes	Mnemonic	Pressed In Idle Mode	Executed In A Program	Learn Mode
x^2	x^2	The display is squared		Programmable
$\sqrt{x^2}$	$\sqrt{x^2}$	Takes square root of displayed number		Programmable
1/x	1/x	Takes the reciprocal of the displayed number		Programmable
π	π	Puts π (3.14159...) into the display		Programmable
PAPER FEED	PF	Advances paper in printer one line at a time		Programmable
X 10^{00}	X 10^{00}	Used to enter exponent or convert a number to scientific notation		Programmable
(	(	Opens a parenthetical entry		Programmable
+–	+/–	Changes sign in display		
)	)	Closes a parenthetical expression, two will always close a statement.		Programmable
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	Used for numerical entry. May be used as subroutine labels.		Programmable
.	.	Decimal point.		Programmable
K	K	Used to access data registers. Register is identified and recalled by digit that follows. Display storage into these registers is accomplished by preceding K and its following digit with = (= K 9, for example).		Programmable
÷	÷	Divide		Programmable
X	X	Multiply		Programmable
–	–	Subtract		Programmable
+	+	Add		Programmable

KEYBOARD SYNOPSIS

Keystrokes	Mnemonic	Pressed In Idle Mode Executed In A Program	Learn Mode
=	=	Equals. Causes accumulation of all entered arithmetic statements. Also causes display storage in a data register when followed by a data register address. (See **STORAGE OPERATIONS AND DATA REGISTERS**.)	Programmable
CLEAR	CLR	Resets calculator to begin again on a new set of operations and data entries.	Programmable
PRINT DISPLAY	PRNT	Print the display (and alpha if entered).	
CLEAR DISPLAY	CD	Clears display to all zeros. Does not alter previously entered operations.	Programmable

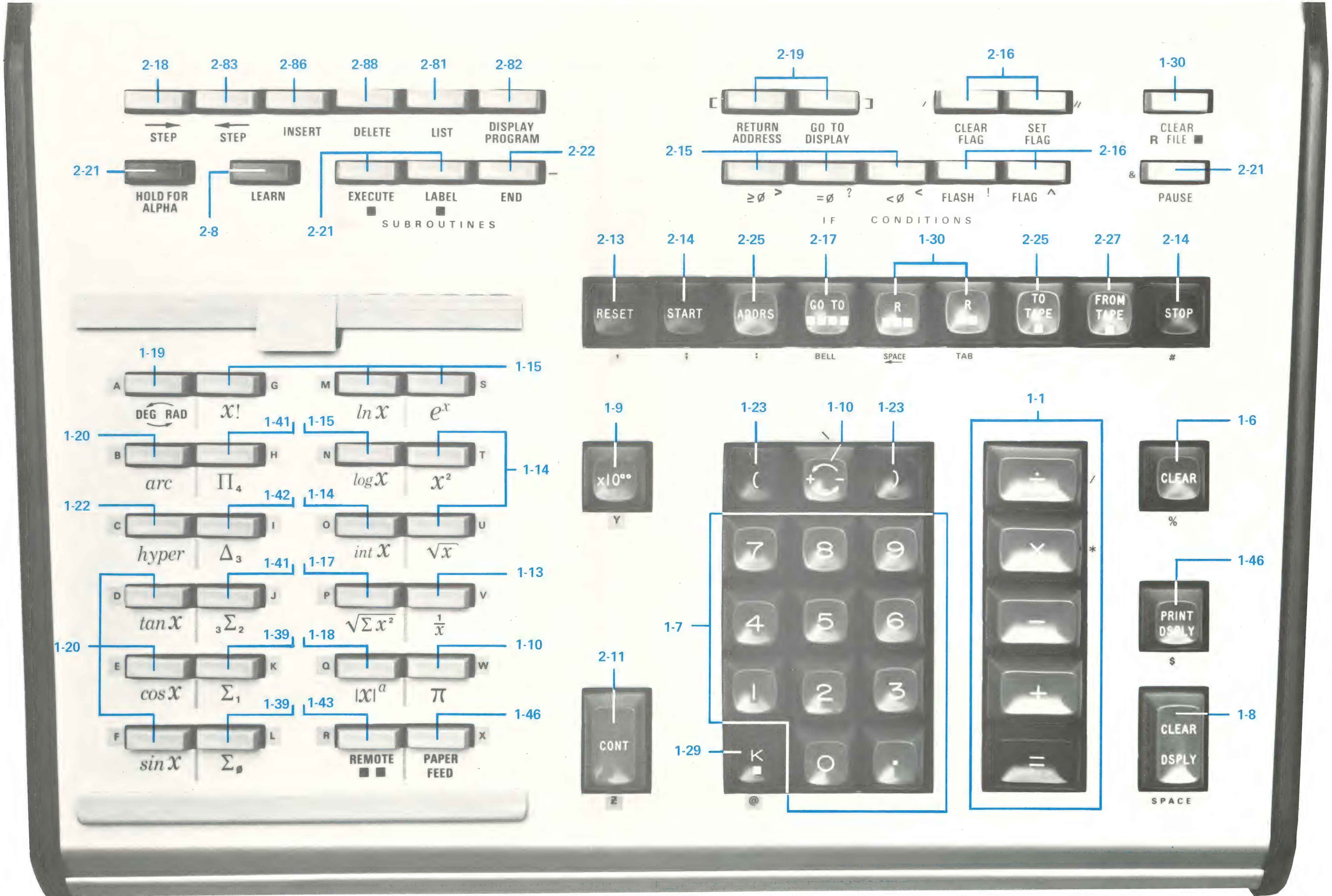
KEYCODES

OCTAL CODE	PRINT OUT	KEY	SYMBOL
001	LBL_	LABEL ■	LBL
002	FTP_	FROM TAPE ■	FTP
003	TTP_	TO TAPE ■	TTP
004	EXC_	EXECUTE ■	EXC
005	CFI_	CLEAR R FILE ■	CFI
007	GOTO	GO TO ■■■■	GT
010	R___	R ■■■	Rxxx
011	R__	R ■■	Rxx
040	CLDP	CLEAR DSPLY	CD
041	IFFL	FLASH	IFFL
042	S FG	SET FLAG	SFG
043	STOP	STOP	STOP
044	PRNT	PRINT DSPLY	PRNT
045	CLR	CLEAR	CLR
046	PAUS	PAUSE	PAUS
047	CLFG	CLEAR FLAG	CLFG
050	(	(	(
051	)	)	)
052	*	×	x
053	+	+	+
054	RSET	RESET	RSET
055	-	—	—
056	.	•	•
057	/	÷	÷
060	0	0	0
061	1	1	1
062	2	2	2
063	3	3	3
064	4	4	4
065	5	5	5
066	6	6	6
067	7	7	7
070	8	8	8
071	9	9	9
072	ADR	ADDRS	ADR
073	STRT	START	STRT
074	IF<0	<ø	IF<0
075	=	=	=
076	IF>=	≥ø	IF≥0
077	IF=0	=ø	IF=0

OCTAL CODE	PRINT OUT	KEY	SYMBOL
100	K_	K ■	K
101	DG/P	DEG RAD	D/R
102	ARC	*arc*	arc
103	HYP	*hyper*	hyp
104	TAN	$tan\,x$	tan
105	COS	$cos\,x$	cos
106	SIN	$sin\,x$	sin
107	X!	$x!$	x!
110	PI4	Π_4	Π_4
111	DLT3	Δ_3	Δ_3
112	3SM2	${}_3\Sigma_2$	${}_3\Sigma_2$
113	SUM1	Σ_1	Σ_1
114	SUM0	Σ_0	Σ_0
115	LN	$ln\,x$	ln
116	LOG	$log\,x$	log
117	INT	$int\,x$	int
120	RSSQ	$\sqrt{\Sigma x^2}$	$\sqrt{\Sigma x^2}$
121	X↑A	$\lvert x\rvert^a$	x^a
122	RM__	REMOTE ■■	RMT
123	E↑X	e^x	e^x
124	X↑2	x^2	x^2
125	SQRT	$\sqrt{x}$	$\sqrt{x}$
126	1/X	$\frac{1}{x}$	1/x
127	PI	π	π
130	PAPK	PAPER FEED	PF
131	*10↑	$\times 10^{00}$	10^{00}
132	CONT	CONT	CONT
133	R AD	RETURN ADDRESS	RADR
134	+/-	+/−	+/—
135	GODP	GO TO DISPLAY	GODP
136	IFFG	FLAG	IFFG
137	ENDR	END	ENDR

OCTAL CODE	PRINT OUT	KEY	SYMBOL
201		LABEL ■	LBL
202		FROM TAPE ■	FTP
203		TO TAPE ■	TTP
204		EXECUTE ■	EXC
205		CLEAR R FILE ■	CFI
207		BELL	BELL
210		SPACE ←	SPC←
211		TAB	TAB
220		← STEP	STP←
221		INSERT	NSRT
222		DELETE	DLT
223		→ STEP	STP→
224		LIST	LIST
225		DISPLAY PROGRAM	DPRG
226		LEARN	LRN
240		SPACE	SPC
241	!	!	!
242	"	"	"
243	#	#	#
244	$	$	$
245	%	%	%
246	&	&	&
247	'	'	'
250	(	(	(
251	)	)	)
252	*	*	*
253	+	+	+
254	,	,	,
255	-	—	—
256	.	.	.
257	/	/	/
260	0	0	0
261	1	1	1
262	2	2	2
263	3	3	3
264	4	4	4
265	5	5	5
266	6	6	6
267	7	7	7
270	8	8	8
271	9	9	9
272	:	:	:
273	;	;	;
274	<	<	<
275	=	=	=
276	>	>	>
277	?	?	?

OCTAL CODE	PRINT OUT	KEY	SYMBOL
300	@	@	@
301	A	A	A
302	B	B	B
303	C	C	C
304	D	D	D
305	E	E	E
306	F	F	F
307	G	G	G
310	H	H	H
311	I	I	I
312	J	J	J
313	K	K	K
314	L	L	L
315	M	M	M
316	N	N	N
317	O	O	O
320	P	P	P
321	Q	Q	Q
322	R	R	R
323	S	S	S
324	T	T	T
325	U	U	U
326	V	V	V
327	W	W	W
330	X	X	X
331	Y	Y	Y
332	Z	Z	Z
333	[	[	[
334	\	\	\
335	]	]	]
336	↑	↑	↑
337	_	_	_



The keyboard location of each key and the page number where it is discussed

INDEX

INDEX

A

B

C

D

INDEX

INDEX

INDEX

MANUAL CHANGE INFORMATION

At Tektronix, we continually strive to keep up with latest electronic developments by adding circuit and component improvements to our instruments as soon as they are developed and tested.

Sometimes, due to printing and shipping requirements, we can't get these changes immediately into printed manuals. Hence, your manual may contain new change information on following pages.

A single change may affect several sections. Sections of the manual are often printed at different times, so some of the information on the change pages may already be in your manual. Since the change information sheets are carried in the manual until ALL changes are permanently entered, some duplication may occur. If no such change pages appear in this section, your manual is correct as printed.

CHANGE INFORMATION

Fill out and return this card for your free subscription to the Tektronix Calculator journal, which talks about new applications, programs, etc.

What additional features would you like to see?

Additional Keys

____________ ____________ ____________

Peripherals

____________ ____________ ____________

Other

____________ ____________ ____________

Which TEKTRONIX product do you currently have?

Tek 21   Tek 31

Are you using TEKTRONIX software?

Statistics   Mathematics   Other (specify) ____________

What additional software would you like to see?

Graphing
Surveying
Mathematical
Educational
Electronic Design
Business Application
General Scientific
Mechanical Engineer
Numerical Control
Other (specify) ____________

What are the most important applications for your calculator?

Consulting
Design/Development
Engineering Support
Information, Data Processing
Maintenance, Service
Management
Manufacturing, Production
Marketing, Sales
Purchasing
Research
Standards, QC
Teaching, Instructional
Other (Specify) ____________

Company or Organization?

Banking
Business Analysis
Chemical
Civil Engineering
Communications
Data Processing
Education
Electronics
Engineering
General Science
Govt., Military
Govt., Non-Military
Insurance
Medical
Metallurgy
Meteorology
Nuclear
Oceanography
Oil
Physics
Steel
Transportation
Utilities
Other (Specify) ____________

Please put me on your list to receive information on new application programs.

Yes   No

Name: ________________________

Title: ________________________

Company: ________________________

City: ____________ State ____________ Zip ________

Comments: ________________________

____

____

FOLD

FIRST CLASS
PERMIT NO. 61
BEAVERTON, OREGON

BUSINESS REPLY MAIL

No postage necessary if mailed in the United States

Postage will be paid by

TEKTRONIX, INC.
P. O. Box 500
Beaverton, Oregon 97005

ATTEN: CALCULATOR MARKETING

FOLD

STAPLE OR TAPE